High-level Synthesis for Domain Specific Computing

Deming Chen

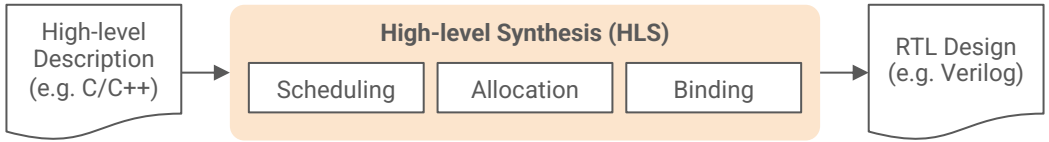
Abel Bliss Professor of Engineering
ECE, University of Illinois at Urbana-Champaign (UIUC)

Co-authors: Hanchen Ye[1], Hyegang Jun[1], Jin Yang[2]

[1]: ECE, UIUC
[2]: Strategic CAD Labs, Intel

Panel on EDA for Domain Specific Computing, ISPD, 2023

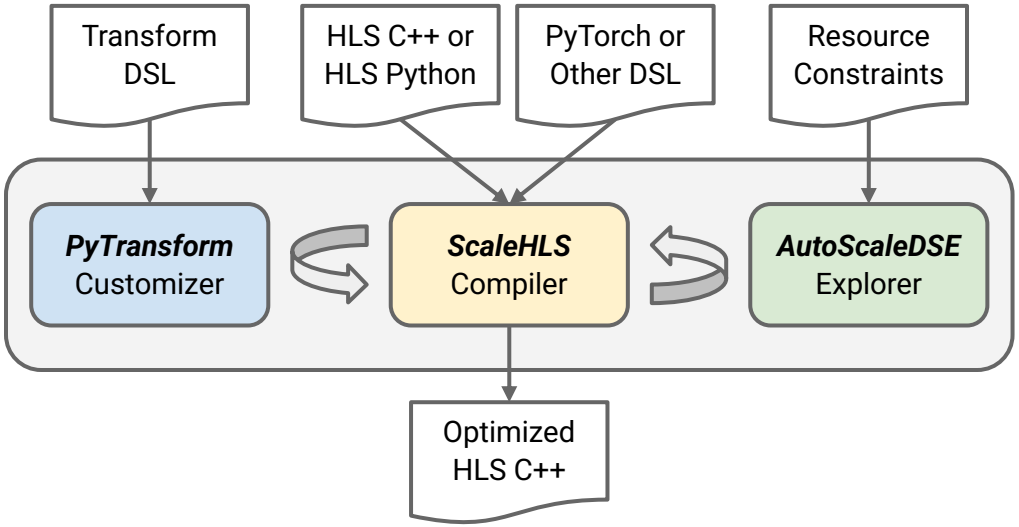
Challenges of Domain-Specific High-level Synthesis (HLS)



- Compilation Challenges
  - Large semantics gap between domain-specific languages and architectures, demanding multi-level modeling and progressive semantics lowering
  - Different HLS optimizations at different abstraction levels
- Exploration Challenges
  - Vast design space enabled by the large amount of tunable design parameters, demanding advanced auto-tuning or design space exploration algorithm
  - HLS design parameters often correlate with each other, further complicating the exploration
- Customization Challenges
  - Domain-specific customized optimizations are essential for the fine-tuning of domain-specific architectures
  - Lack high-level and expressive programming interface for designers to define domain-specific customizations

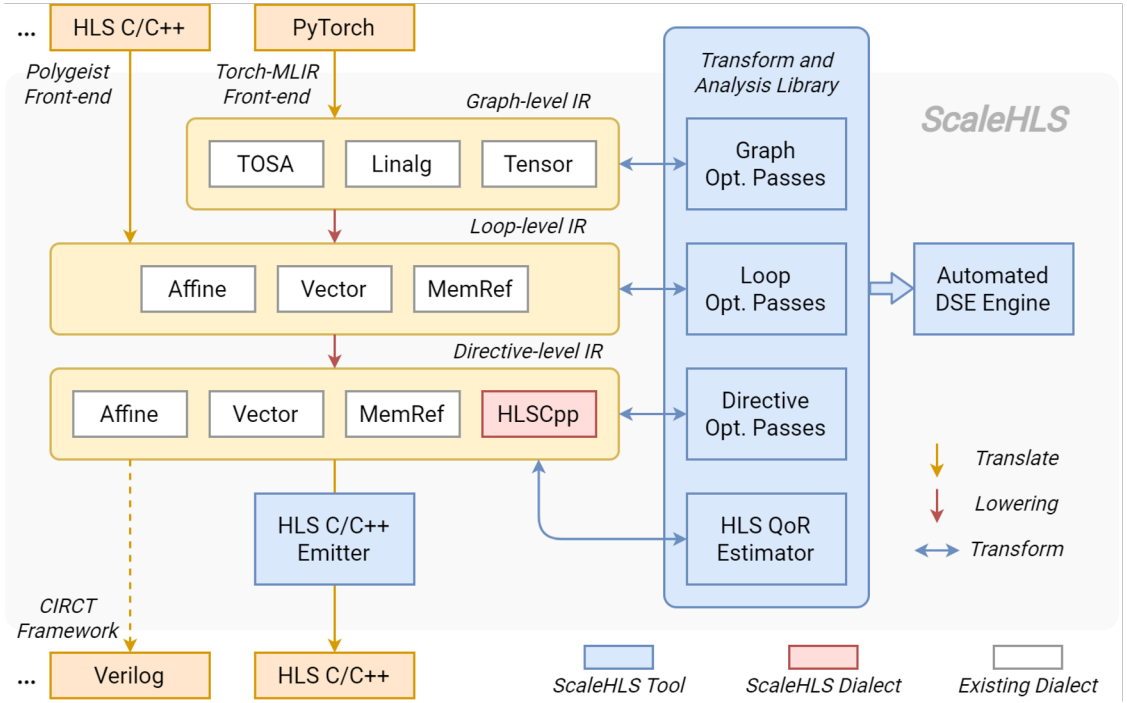
Our Approach - A Holistic Solution

- *ScaleHLS* Compiler
  - Multi-level representation and optimization for HLS
  - Source-to-source compilation
- *AutoScaleDSE* Explorer
  - ML-aided exploration engine for Pareto design points in the performance-area domain
- *PyTransform* Customizer
  - A pattern-driven Python-based DSL to define customized optimization patterns



[1] H. Ye, et al., "ScaleHLS: A New Scalable High-Level Synthesis Framework on Multi-Level Intermediate Representation", HPCA, 2022.
[2] H. Ye, et al., "Invited: ScaleHLS, a Scalable High-Level Synthesis Framework with Multi-level Transformations and Optimizations", DAC, 2022.
[3] H. Jun, et al., "AutoScaleDSE: A Scalable Design Space Exploration Engine for High-Level Synthesis", TRETS, 2022.
[4] H. Ye, et al., "High-level Synthesis for Domain Specific Computing", ISPD, 2023.

ScaleHLS: HLS + MLIR for the First Time!



Inputs

C/C++ Polygeist [1]

PyTorch Torch-MLIR [2]

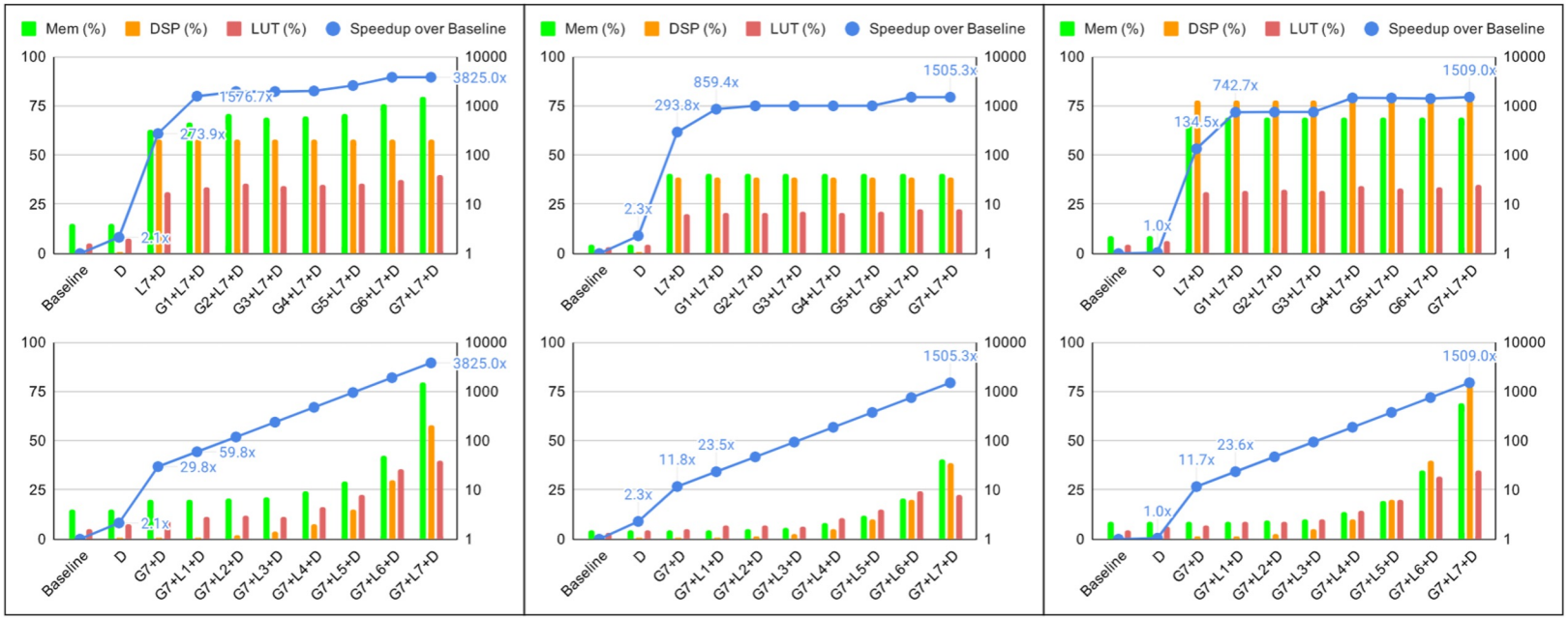
Outputs

Vitis HLS C/C++
C/C++ Emitter

Verilog CIRCT [3]
(work-in-progress)

[1] Polygeist: https://github.com/wsmoses/Polygeist [2] Torch-MLIR: https://github.com/llvm/torch-mlir [3] CIRCT: https://github.com/llvm/circt

Evaluation of ScaleHLS with PyTorch: Ablation Study



(a) ResNet-18 **(b)** VGG-16 **(c)** MobileNet

D, *L{n}*, and *G{n}* denote directive, loop, and graph optimizations, respectively. Larger *n* indicates larger loop unrolling factor and finer dataflow granularity for loop and graph optimizations, respectively.

ScaleHLS is Open-Sourced

ScaleHLS GitHub Repository

https://github.com/hanchenye/scalehls

Since 2022, ScaleHLS has around

- ✓ **25,000 views**
- ✓ **2,400 downloads** 🔥🔥

For HLS Researchers

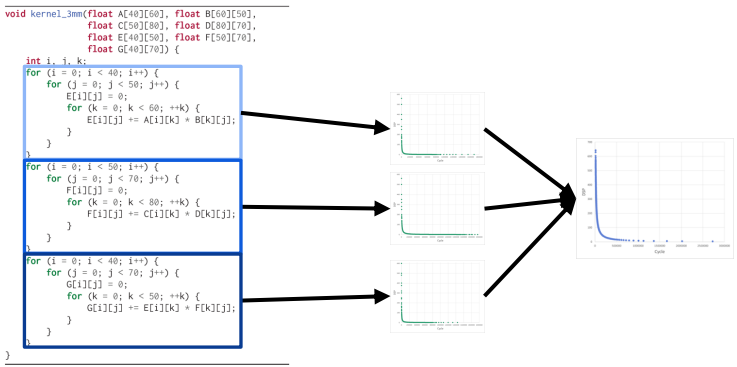
1. Rapidly implement new HLS optimization algorithms on top of the multi-level IR
2. Investigate new DSE algorithms using the transform and analysis library
3. Rapidly build an end-to-end HLS optimization flow and demonstrate your awesome works!

For HLS Users

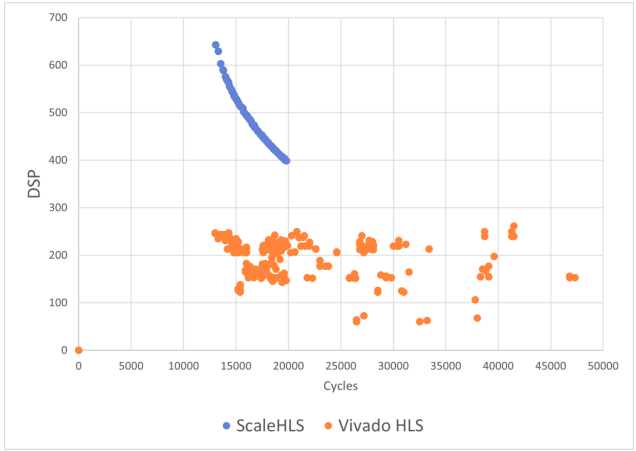
1. Optimize HLS designs using the multi-level optimization passes
2. Avoid premature design choices by using the QoR estimator to estimate the latency and utilization
3. Find optimized HLS designs with the automated DSE engine

H. Ye, et al., “ScaleHLS: A New Scalable High-Level Synthesis Framework on Multi-Level Intermediate Representation”, HPCA, 2022.
H. Ye, et al., “Invited: ScaleHLS, a Scalable High-level Synthesis Framework with Multi-level Transformations and Optimizations”, DAC, 2022.

AutoScaleDSE: Motivation



Divide and Conquer the Design Space

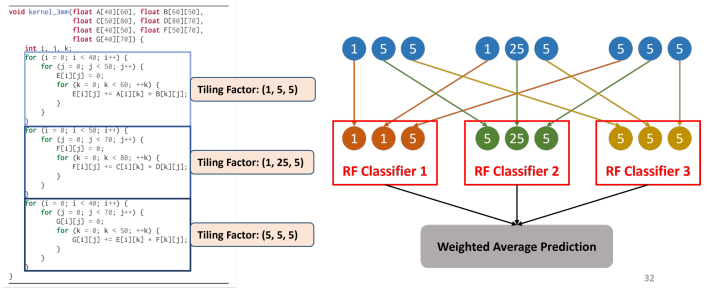


The Global Design Space

- **Vast Design Space:** HLS design space exploration has scalability issues due to the vast design space.
- **Divide and Conquer:** An effective approach to deal with large designs, by working with sub-designs.
- **Merging Problem:** Due to interdependencies, merged sub-designs may have an adverse effect.
- **Non-Pareto Global Design Space:** 40% - are Pareto Optimal, 60% not Pareto Optimal.
- **Why Sub-designs are Compatible/Incompatible:** Interaction between the heuristics of the HLS compiler

H. Jun, et al., "AutoScaleDSE: A Scalable Design Space Exploration Engine for High-Level Synthesis", TRETS, 2022.

AutoScaleDSE: Idea + Results



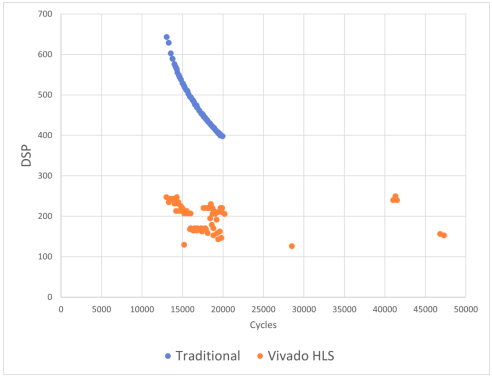
Using the Random Forest Classifier

	No. Designs	Prob. Sizes	Implementation	Latency	Speedup	DSP	FF	LUT	GA
bicg	2	390, 410	Baseline	16 *ms*	1x	0%	0%	0%	No
			ScaleHLS	42.050 *µs*	380x	50%	24%	37%	No
			AutoScaleDSE	42.050 *µs*	380x	50%	24%	37%	No
correlation	4	240, 260	Baseline	1.367 *s*	1x	0%	0%	1%	No
			ScaleHLS	0.942 *s*	1.5x	15%	18%	51%	No
			AutoScaleDSE	15.467 *ms*	88x	4%	7%	27%	No
2mm	2	180, 190, 210, 220	Baseline	1.542 *s*	1x	0%	0%	0%	No
			ScaleHLS	8.281 *ms*	186x	55%	21%	46%	No
			AutoScaleDSE	1.722 *ms*	895x	76%	17%	46%	No
3mm	3	180, 190, 200, 210, 220	Baseline	2.054 *s*	1x	0%	0%	0%	No
			ScaleHLS	72.930 *ms*	28x	35%	20%	33%	No
			AutoScaleDSE	1.996 *ms*	1029x	61%	17%	37%	No

Table 8. AutoScaleDSE vs ScaleHLS DSE using the Polybench [29] medium dataset.

Evaluation Results

- **Concise Representation of Design Points:** The Tiling strategy is used to capture the desired parallelization.
- **Random Forest Classifier:** Predict Sub-design compatibility using the RF Classifier -> 97% accuracy.
- **New Global Design Space:** Drastically decrease the number of non-Pareto optimal points.
- **Results:** Was able to beat ScaleHLS by up to **59x in speed.**



New Global Design Space

PyTransform: Montgomery Reduction as an Example

- Montgomery algorithm is a method for performing fast modular multiplication (*ab mod M*), which is widely used in cryptography algorithm
- Montgomery reduction (aR^{-1} *mod M*) is the key component of Montgomery algorithm

```
def montgomery_redc(A, M, v, k, z):
    C = []
    for i in range(0, A.shape[0]):
        a = A[i]
        s = (a * v) & z
        r = (a + s * M) >> k
        C[i] = r if r < M else r - M
    return C
```

Montgomery Reduction in Python

```
s = (a * v) & z
```

Expression Rewrite*

```
s = (a * -1) mod S
  = -a mod S
  = S - (a mod S)
  = S - (a & z)
```

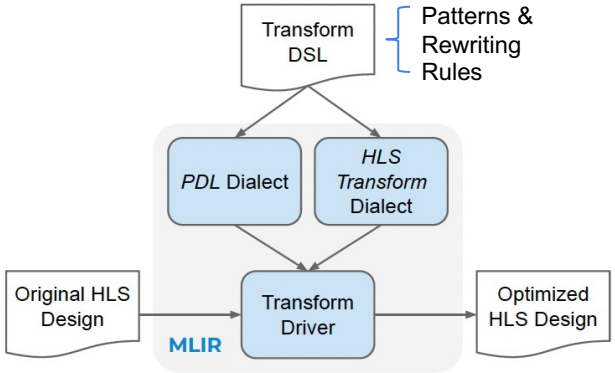
Reduce Area ✓
3 DSPs + 52 LUTs => 71 LUTs
***or* 1085 LUTs => 71 LUTs**

Hard to Automate ✗
>100 lines of C++ code in the MLIR compiler

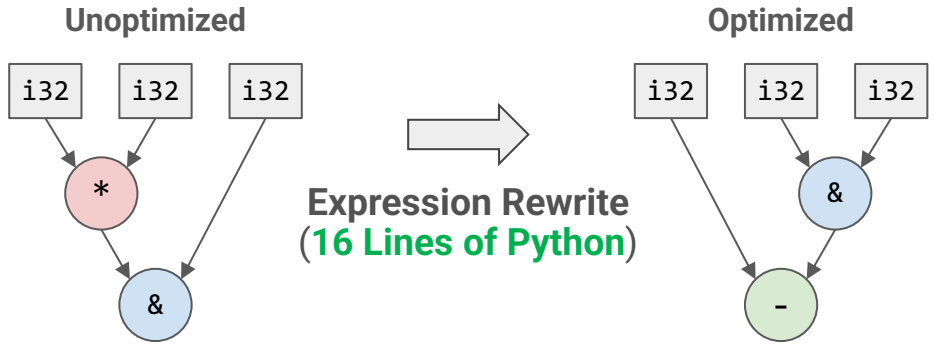
Low-Level Implement ✗
Require **low-level compilation expertise**

*Require: `S = z + 1; v == -1; S is power of 2`

PyTransform - Defining & Automating Customizations



Architecture of PyTransform flow

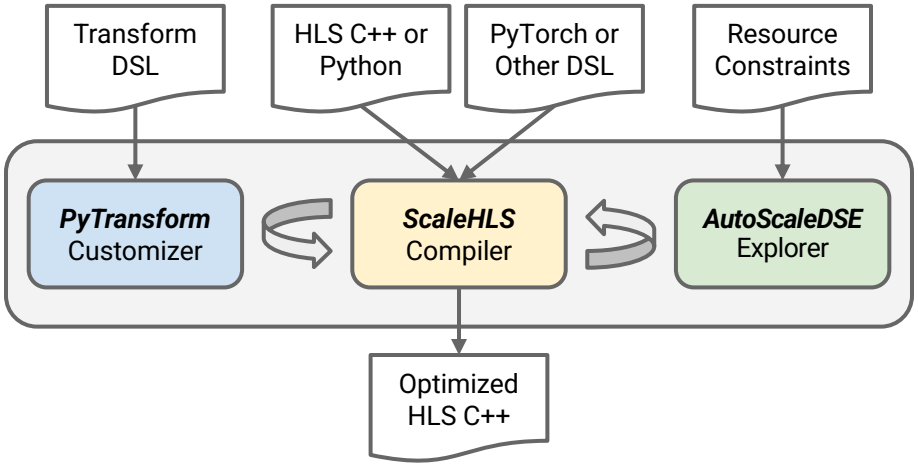


- PyTransform optimization results on Montgomery reduction.
- 6 customizations defined in PyTransform, including the rewriting: (a * v) & z => S - (a & z).

Implementation	Latency	Latency Compare	DSP	DSP Compare
Original	262	1.00×	192	1.00×
PyTransform	265	1.01×	130	0.68×

H. Ye, et al., "High-level Synthesis for Domain Specific Computing", ISPD, 2023.

Conclusions & Future Work



- Optimization of dataflow architecture
- Optimization of external memory access
- Dataflow-aware design space exploration
- Verifiable HLS
- Low-latency domain-specific applications