ScaleFlow: Scalable High-Level Synthesis for Large Dataflow Applications

Hanchen Ye, Hyegang Jun, and Deming Chen *University of Illinois at Urbana-Champaign*

ILLINOIS

Motivation - Goal

- Automatically generate **efficient task-level pipeline (dataflow)** in large-scale HLS designs

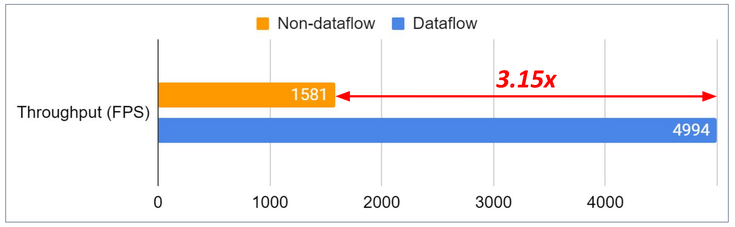


Fig. 1: Throughput of LeNet on PYNQ-Z2 FPGA

Utilization (%)	LUT	BRAM	DSP
Non-dataflow	*65.40%*	98.93%	100.00%
Dataflow	*67.90%*	98.93%	100.00%

Table. 1: Resource Utilization of LeNet on PYNQ-Z2 FPGA

Motivation - Challenges

- The gap between serial and dataflow programming model is huge
  - It's difficult for designers to construct **legal and efficient** dataflow structure manually
- The design space of dataflow is huge
  - It's difficult for designers to reason the optimal on-chip buffer sizes, parallelization factors, streaming strategies, etc.

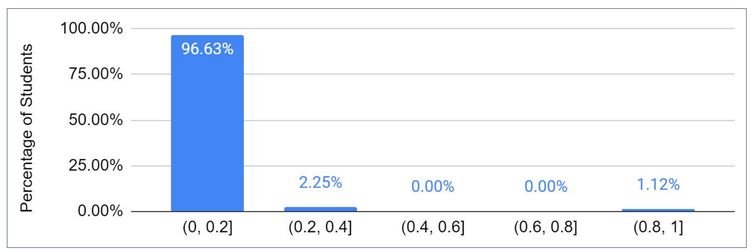


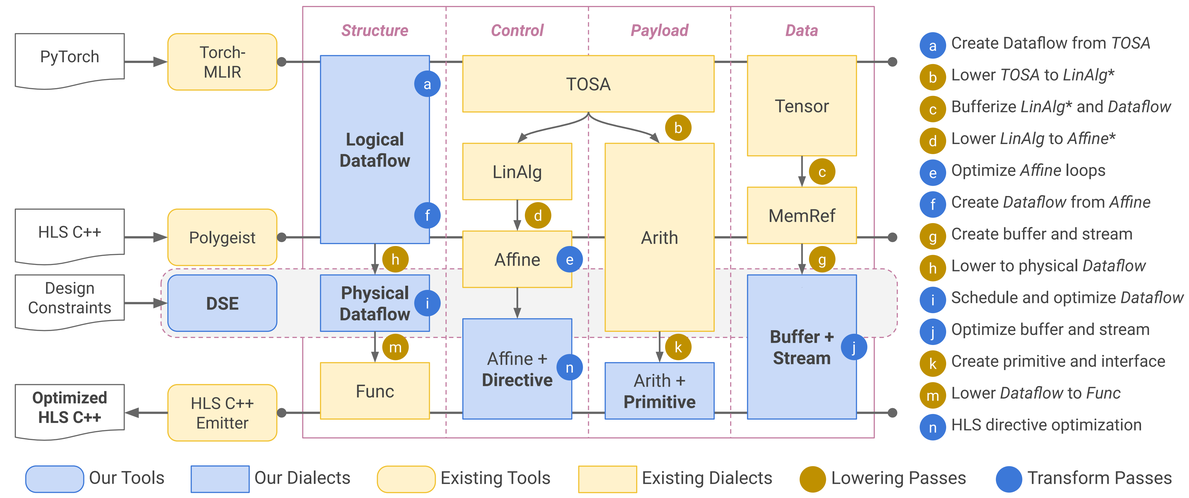
Fig. 2: Students are requested to accelerate LeNet on PYNQ-Z2 using HLS. The figure shows the percentage of students' submissions (Y axis) in each performance range (X axis). The performances are normalized with respect to expert design's performance with dataflow (4994 FPS).

ScaleHLS, the previous version of ScaleFlow, is published at HPCA'22 and DAC'22 and **open-sourced** on GitHub:

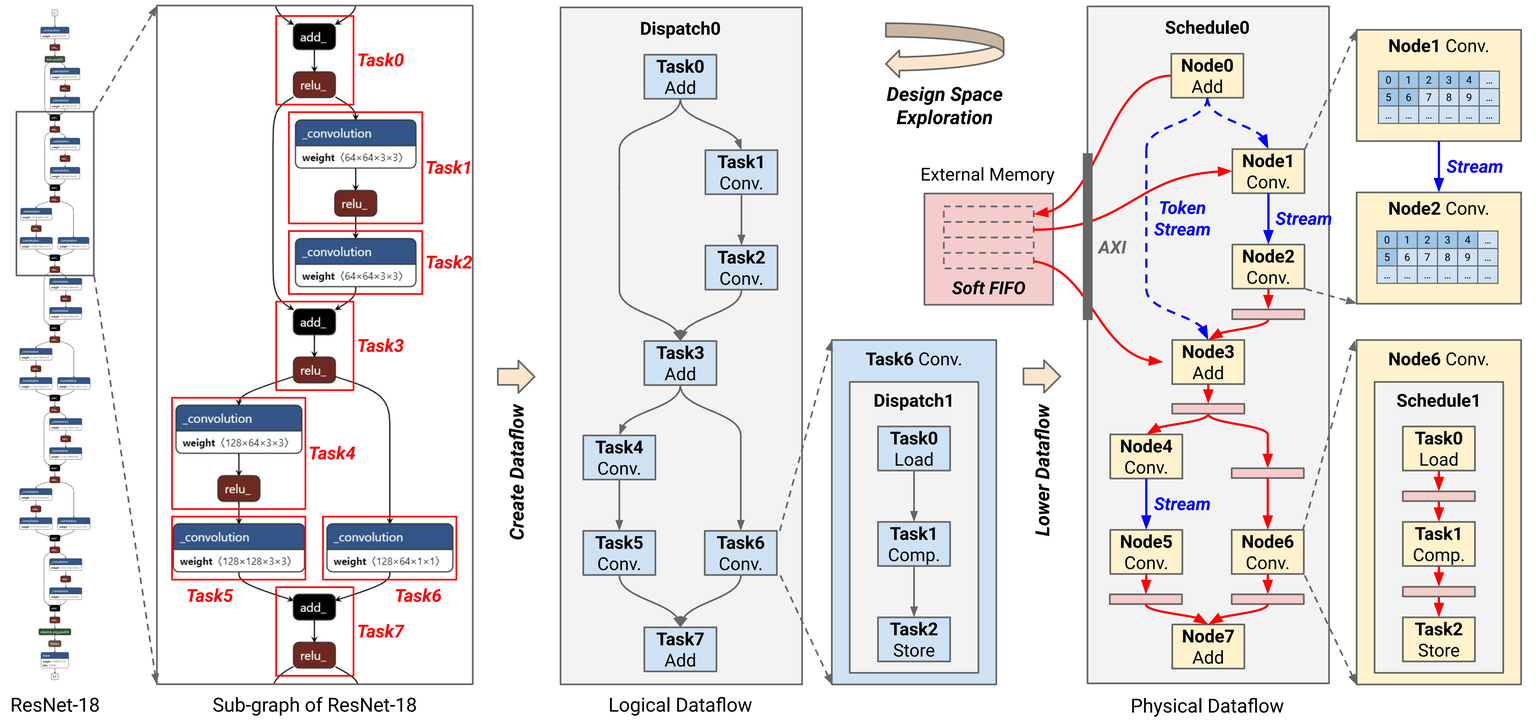
ScaleHLS GitHub Repository
https://github.com/hanchenye/scalehls

MLIR-based ScaleFlow Framework

- Built on top of the **MLIR** compiler infrastructure [1]
- Take PyTorch model or HLS C++ as inputs and generate optimized dataflow designs in HLS C++
- **Logical dataflow:**
  - High-level dataflow representation
  - Used for dataflow structure generation and task dispatch
- **Physical dataflow:**
  - Explicit shared-buffer and stream channel representation
  - Used for dataflow scheduling and optimization



ResNet-18 Compilation Walkthrough



- Start from the computation graph compiled from PyTorch model
- At logical dataflow level, the **hierarchical** dataflow structure is created from the original computation graph
- At physical dataflow level, shared-buffers and stream channels are created and optimized. The parallelization factors of all nodes are balanced according to their computational complexity.