Semiconductor
Research
Corporation

ScaleFlow: High-Level Synthesis for Large Dataflow Applications

Hanchen Ye, Deming Chen
University of Illinois at Urbana-Champaign
hanchen8@illinois.edu, dchen@illinois.edu

Technology Transfer

Industry Interactions

- Jin Yang (Intel)

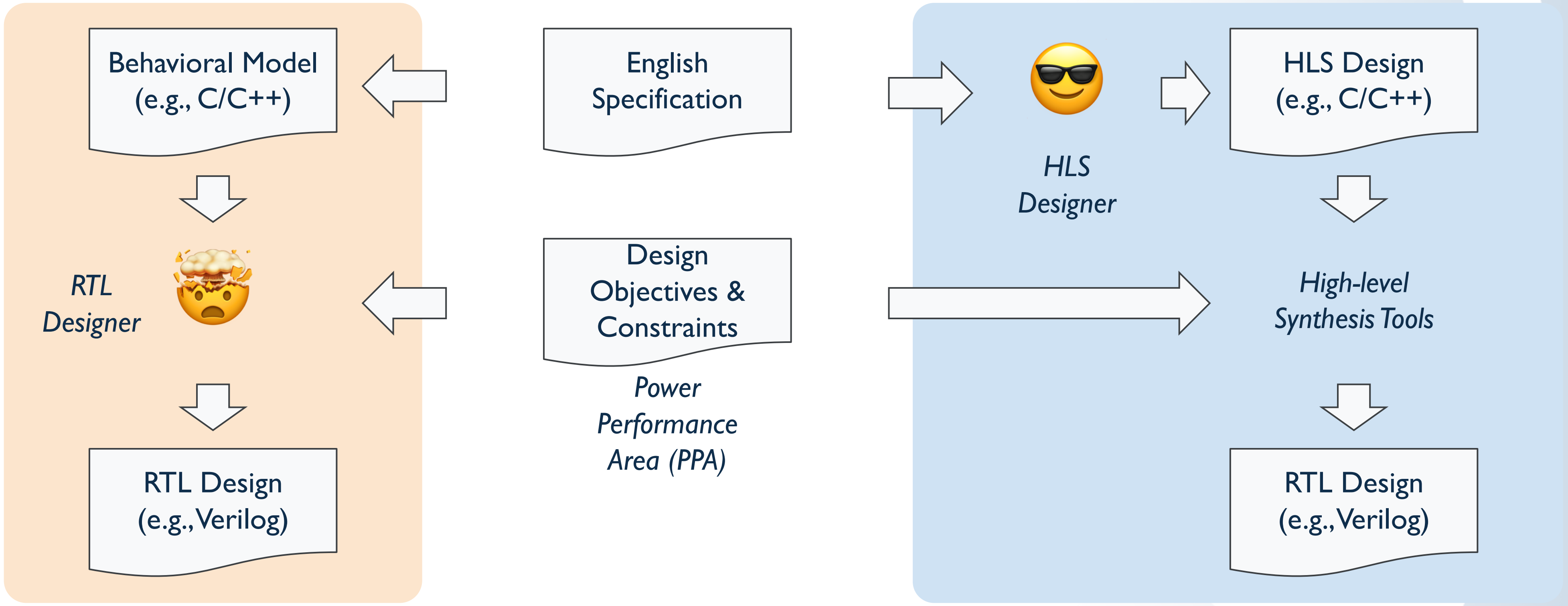
Internships

- Hanchen Ye (Intel Labs, 2022)

Publications/presentations

- High-Level Synthesis for Domain Specific Computing, ISPD'23 (Invited).
- HIDA: A Hierarchical Dataflow Compiler for High-Level Synthesis, ASPLOS'24 (To appear).

Background: High-level Synthesis (HLS)


Behavioral Model (e.g., C/C++)
RTL Designer
RTL Design (e.g., Verilog)
English Specification
Design Objectives & Constraints
Power Performance Area (PPA)
HLS Designer
HLS Design (e.g., C/C++)
High-level Synthesis Tools
RTL Design (e.g., Verilog)

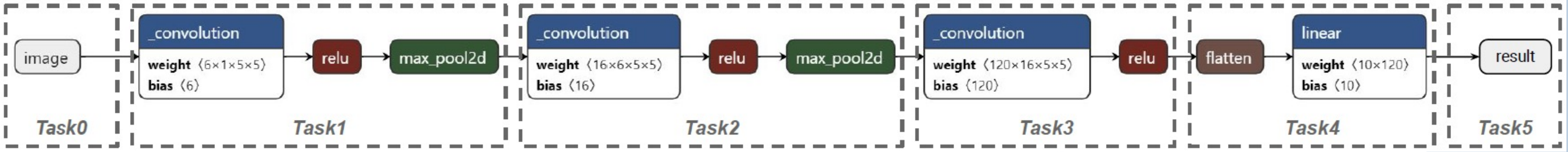
RTL Design Flow

- **Manual** optimization and scheduling
- **Long** design cycle
- **Low** portability against different PDK or PPA requirements

HLS Design Flow

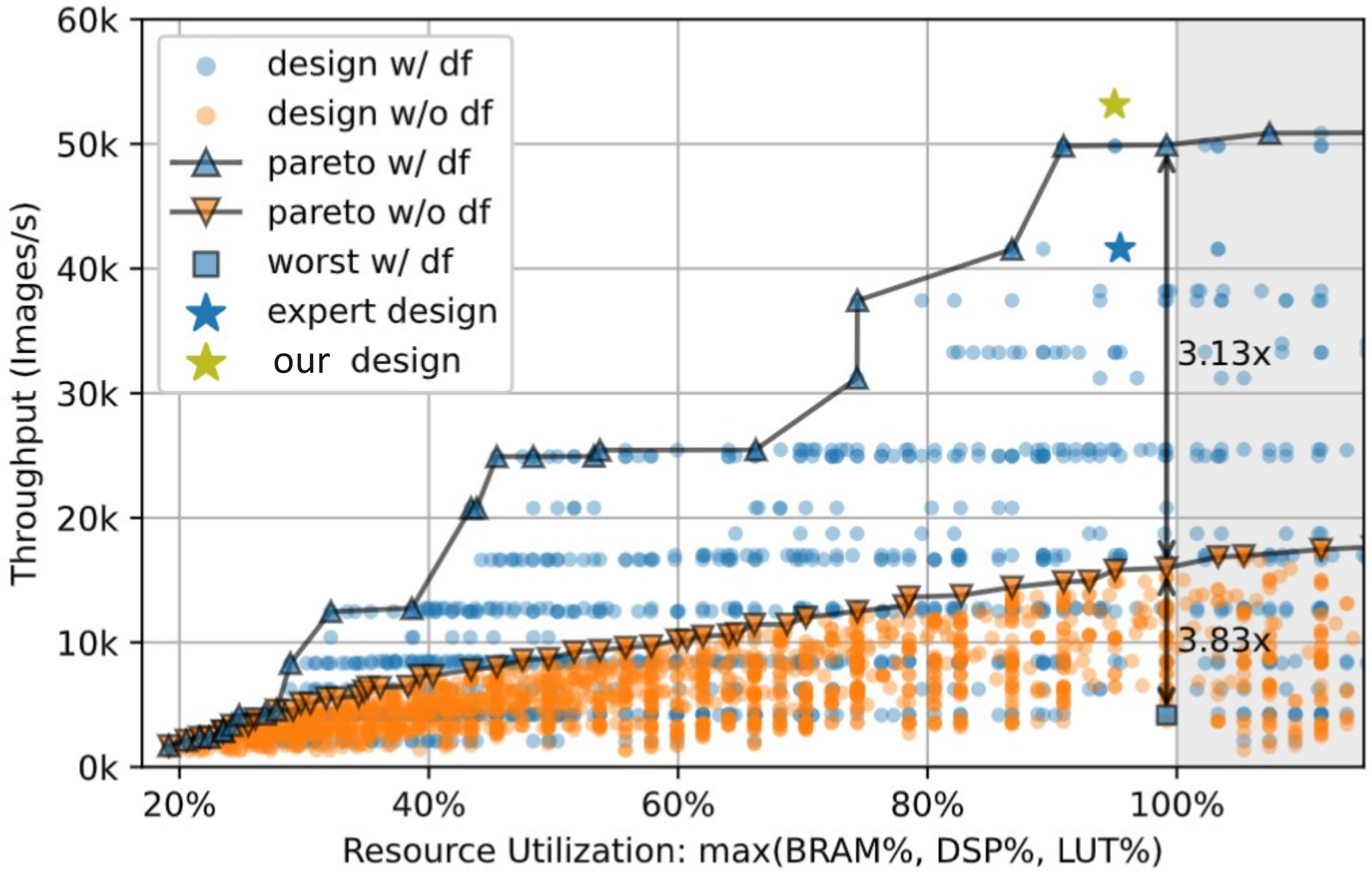
- **Automated** optimization and scheduling
- **Short** design cycle
- **High** portability against different PDK or PPA requirements

Motivation: LeNet as Example



1. Rewrite PyTorch model to C++
2. Layer fusion and layer parallelization
   - Fusion strategy, parallel factor, etc.
3. Enable coarse-grained **dataflow**
   - Inter-task communication, external memory access, etc.
4. Tune design parameters
   - Iterate with downstream tools (2-10 minutes per iteration)

Motivation: LeNet as Example (Cont'd)

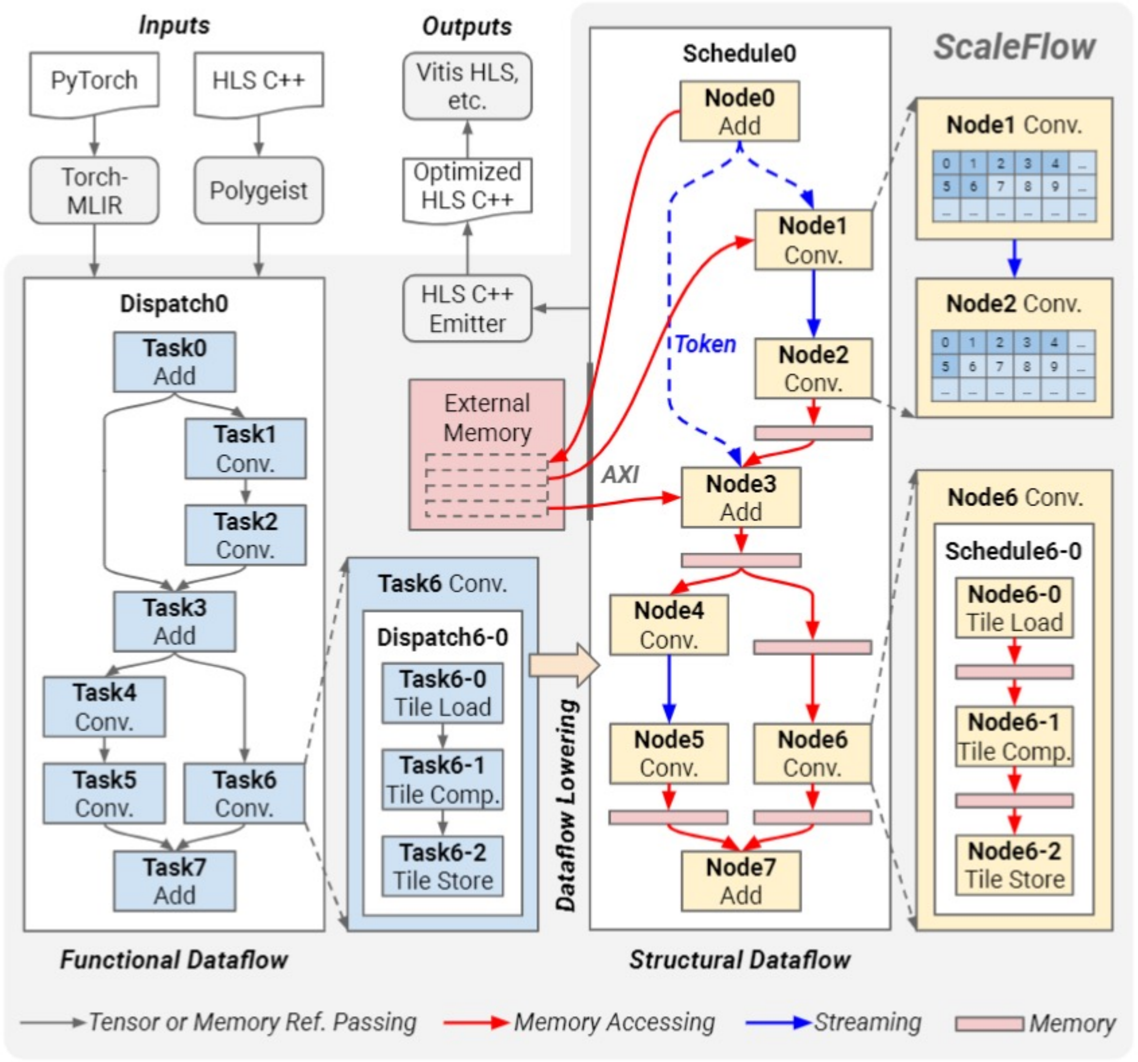


- Dataflow designs are Pareto-dominating
- Dataflow cannot guarantee a good trade-off
- Dataflow design space is vast and difficult to comprehend
  - The design space contains more than 2.4×10^4 points
- Automated tool outperforms exhaustive search

Productivity - Performance - Scalability

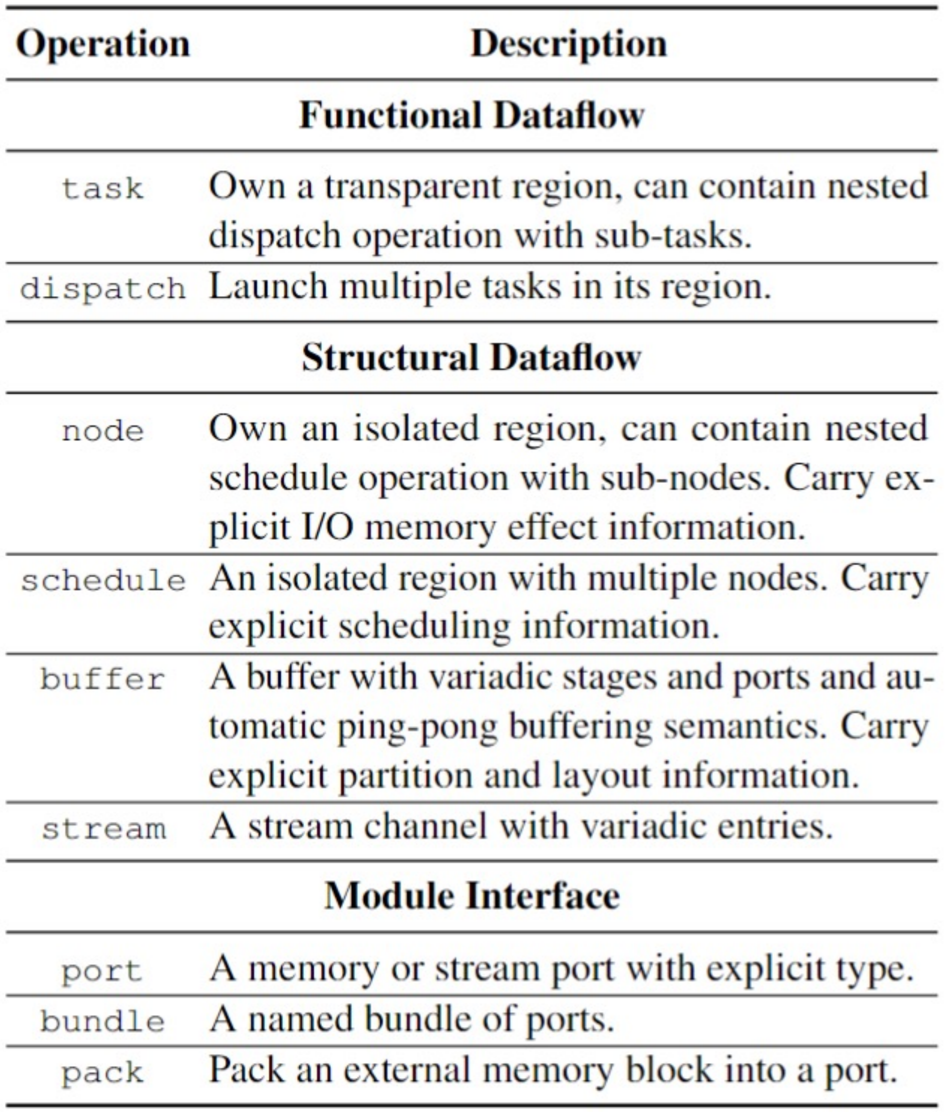
SRC

ScaleFlow Framework



- PyTorch or C/C++ as input
- Optimized C++ dataflow design as output
- Two-level dataflow representation
  - **Functional** dataflow
  - **Structural** dataflow
- Decoupled functional and structural dataflow optimization

ScaleFlow Intermediate Representation

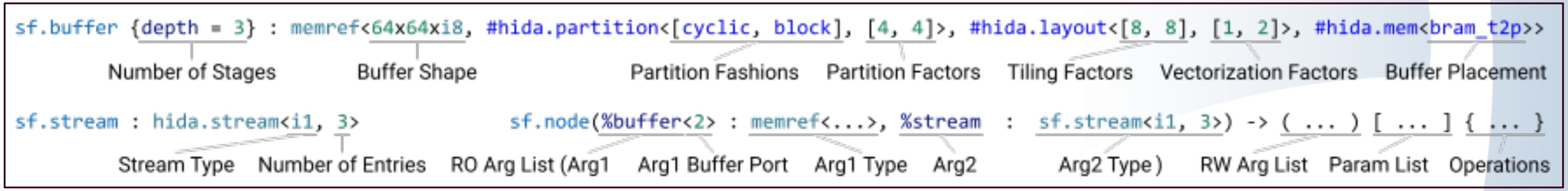


Operation	Description
Functional Dataflow	
`task`	Own a transparent region, can contain nested dispatch operation with sub-tasks.
`dispatch`	Launch multiple tasks in its region.
Structural Dataflow	
`node`	Own an isolated region, can contain nested schedule operation with sub-nodes. Carry explicit I/O memory effect information.
`schedule`	An isolated region with multiple nodes. Carry explicit scheduling information.
`buffer`	A buffer with variadic stages and ports and automatic ping-pong buffering semantics. Carry explicit partition and layout information.
`stream`	A stream channel with variadic entries.
Module Interface	
`port`	A memory or stream port with explicit type.
`bundle`	A named bundle of ports.
`pack`	Pack an external memory block into a port.

- Functional dataflow
  - Designed for efficient dataflow manipulation, such as task fusion
  - Support both tensor and buffer semantics
- Structural dataflow
  - Designed for low-level micro-architectural optimizations
  - Explicit communication modeling
- Module interface
  - Designed to model external memory accesses

SRC

ScaleFlow Structural Dataflow

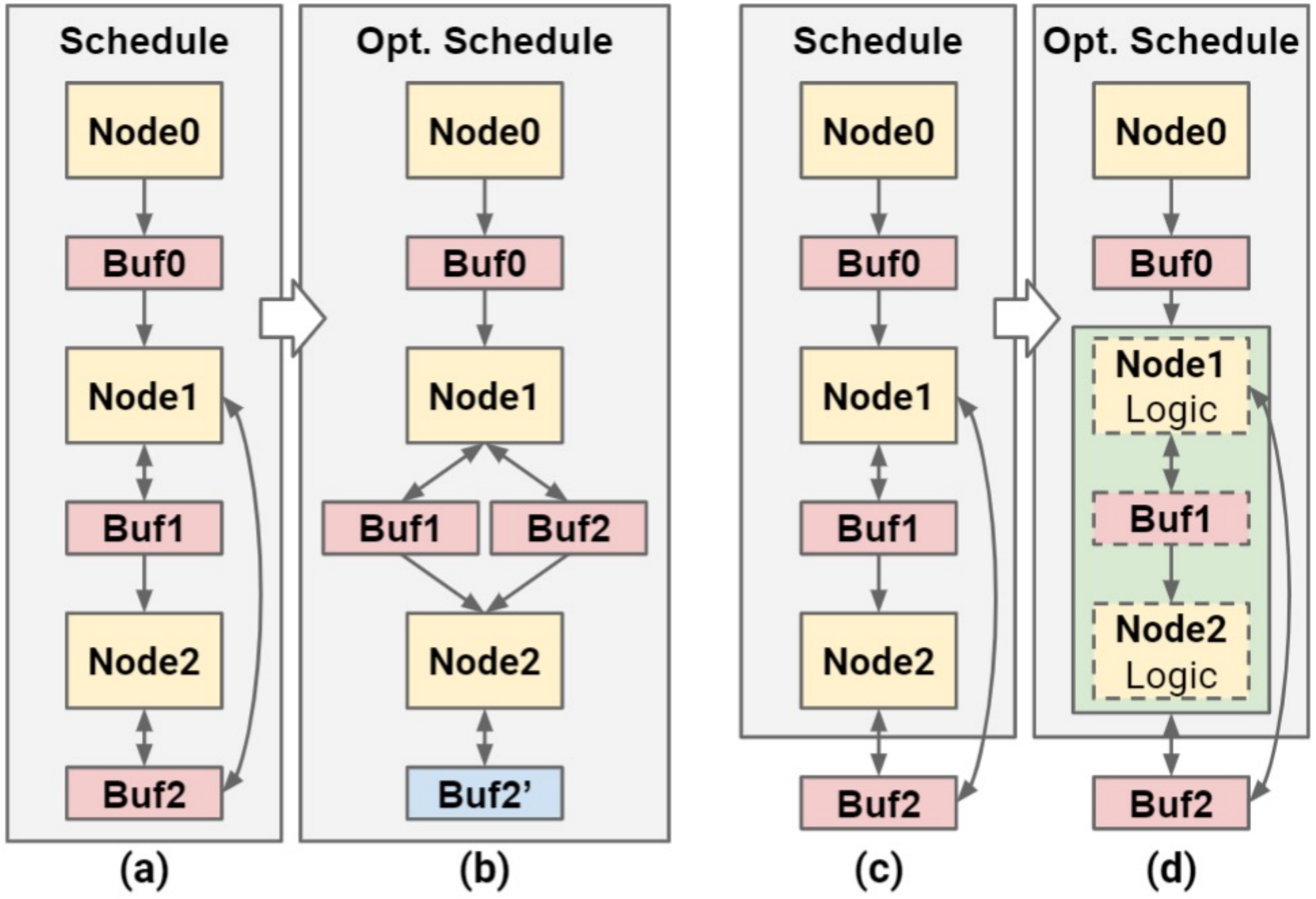


```
sf.buffer {depth = 3} : memref<64x64xi8, #hida.partition<[cyclic, block], [4, 4]>, #hida.layout<[8, 8], [1, 2]>, #hida.mem<bram_t2p>>
sf.stream : hida.stream<i1, 3>        sf.node(%buffer<2> : memref<...>, %stream  :  sf.stream<i1, 3>) -> ( ... ) [ ... ] { ... }
```

- Memory-Mapped and stream buffer representation
  - Explicit partition fashion, partition factors, tiling factors, vectorization factors, placement, etc.
  - Inherent ping-pong buffering semantics
- Isolated dataflow node representation
  - Isolated from context above
  - Explicit memory effect annotation: inputs, outputs, and parameters.

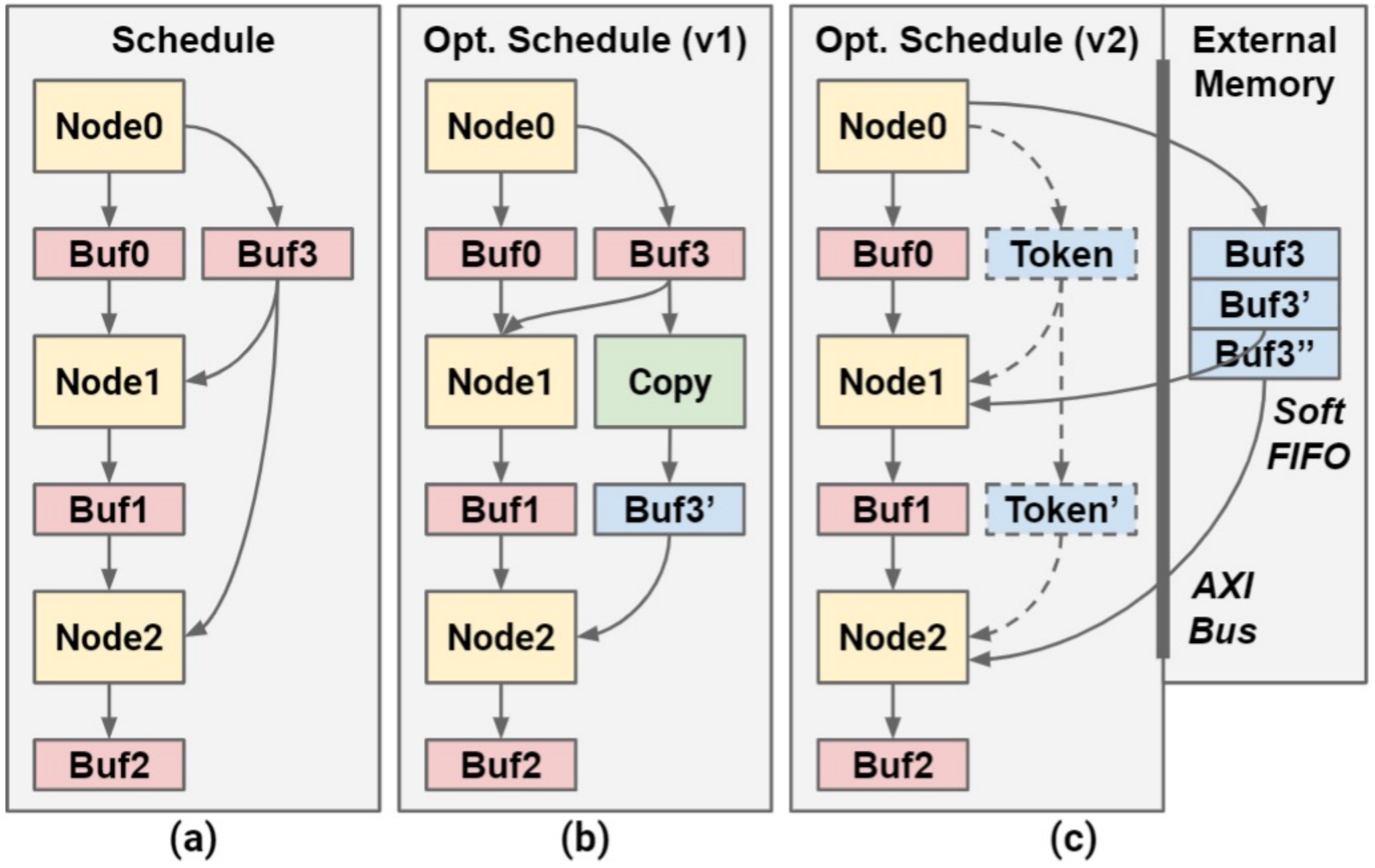
SRC

ScaleFlow Optimization: Multi-producer Elimination



- Multiple producers writing the same buffer simultaneously can introduce data racing
- Solution 1: Buffer duplication
  - (a) => (b)
  - For buffers without external memory effects
- Solution 2: Node fusion
  - (c) => (d)
  - For external buffers

ScaleFlow Optimization: Datapath Balancing



- Imbalanced datapaths can degrade the overall pipeline performance
- Solution 1: On-chip buffer duplication
  - (a) => (b)
  - Balance datapath with copy node insertion
- Solution 2: Soft FIFO in external memory
  - (a) => (c)
  - Token FIFOs are also generated to maintain the dependencies

SRC

ScaleFlow Optimization: Dataflow Parallelization

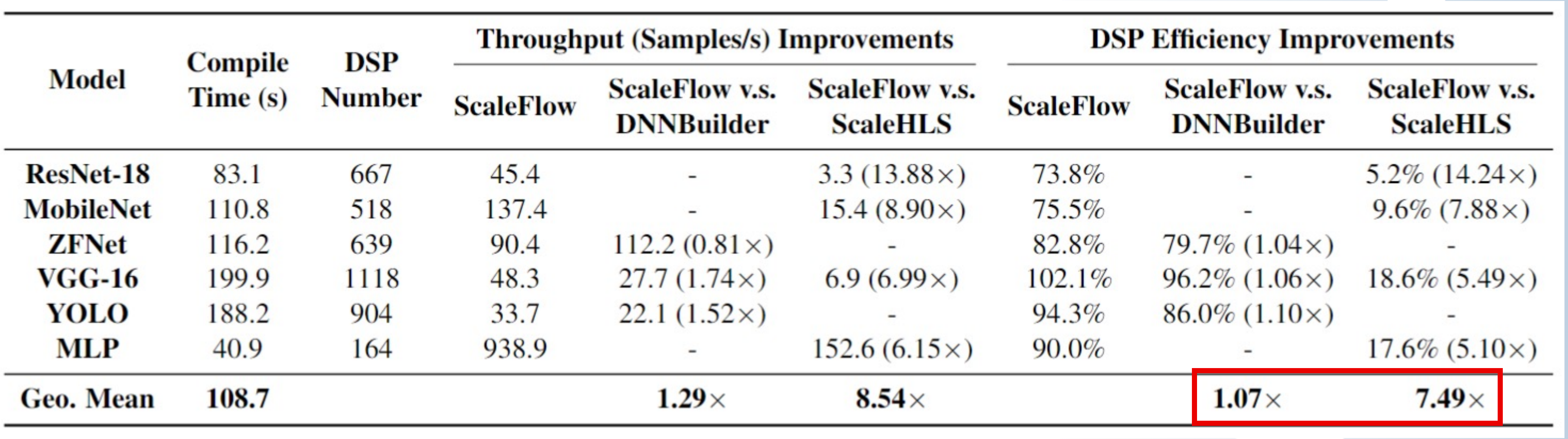
```
float A[32][16];
NODE0_I: for (int i=0; i<32; i++)
  NODE0_K: for (int k=0; k<16; k++)
    A[i][k] = ...; // Load array A.

float B[16][16];
NODE1_K: for (int k=0; k<16; k++)
  NODE1_J: for (int j=0; j<16; j++)
    B[k][j] = ...; // Load array B.

float C[16][16];
NODE2_I: for (int i=0; i<16; i++)
  NODE2_J: for (int j=0; j<16; j++)
    NODE2_K: for (int k=0; k<16; k++)
      C[i][j] = A[i*2][k] * B[k][j];
```

- For dataflow architecture, the local optimal schedule cannot automatically lead the global optimality
  - Memory data layout
  - Connectedness of nodes
  - Computation intensity of nodes
- Intensity and connection-aware dataflow parallelization
  - Constrain the local design space exploration with “intensity” and “connection” of nodes

PyTorch Evaluation



Model	Compile Time (s)	DSP Number	Throughput (Samples/s) Improvements			DSP Efficiency Improvements		
			ScaleFlow	ScaleFlow v.s. DNNBuilder	ScaleFlow v.s. ScaleHLS	ScaleFlow	ScaleFlow v.s. DNNBuilder	ScaleFlow v.s. ScaleHLS
ResNet-18	83.1	667	45.4	-	3.3 (13.88×)	73.8%	-	5.2% (14.24×)
MobileNet	110.8	518	137.4	-	15.4 (8.90×)	75.5%	-	9.6% (7.88×)
ZFNet	116.2	639	90.4	112.2 (0.81×)	-	82.8%	79.7% (1.04×)	-
VGG-16	199.9	1118	48.3	27.7 (1.74×)	6.9 (6.99×)	102.1%	96.2% (1.06×)	18.6% (5.49×)
YOLO	188.2	904	33.7	22.1 (1.52×)	-	94.3%	86.0% (1.10×)	-
MLP	40.9	164	938.9	-	152.6 (6.15×)	90.0%	-	17.6% (5.10×)
Geo. Mean	**108.7**			**1.29**×	**8.54**×		**1.07**×	**7.49**×

Conclusion

- A two-level dataflow intermediate representation, named Functional and Structural dataflow, for the optimization and code generation of dataflow architectures.
- A multi-level dataflow optimization pipeline, including multi-producer elimination, datapath balancing, dataflow parallelization, etc.
- Neural network evaluation on FPGA shows 1.07x and 7.49x higher computational efficiency compared to dedicated RTL accelerator and the SOTA HLS optimization flow.