HIDA: Hierarchical Dataflow Compiler for High-Level Synthesis

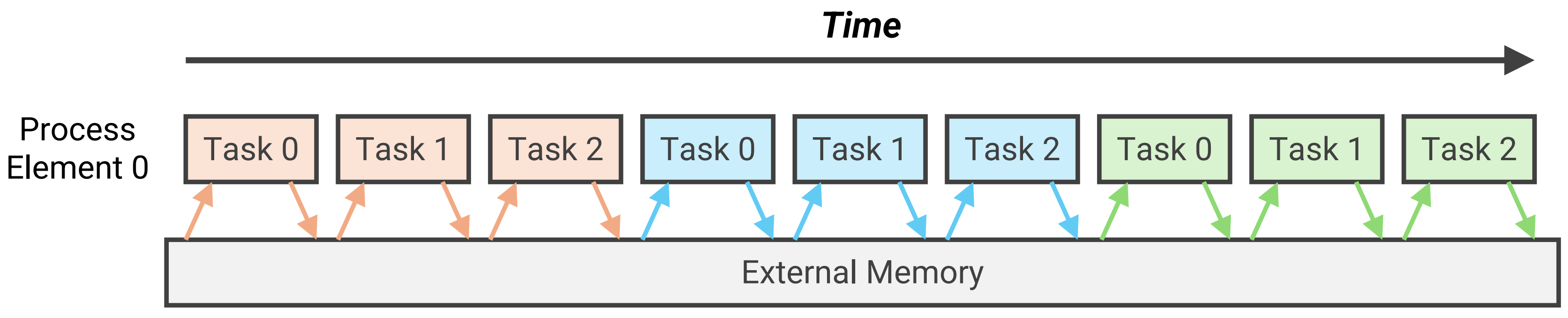
Hanchen Ye, Hyegang Jun, Deming Chen

University of Illinois at Urbana-Champaign

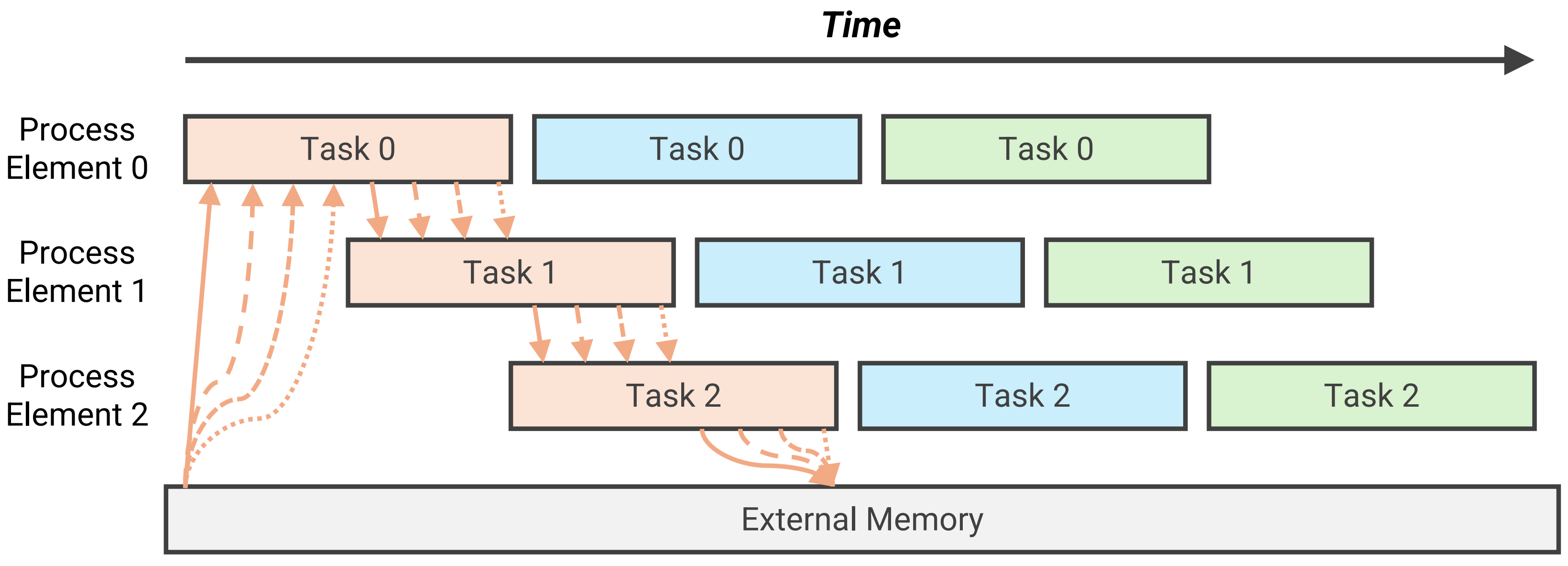
UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

ASPLOS

Motivation

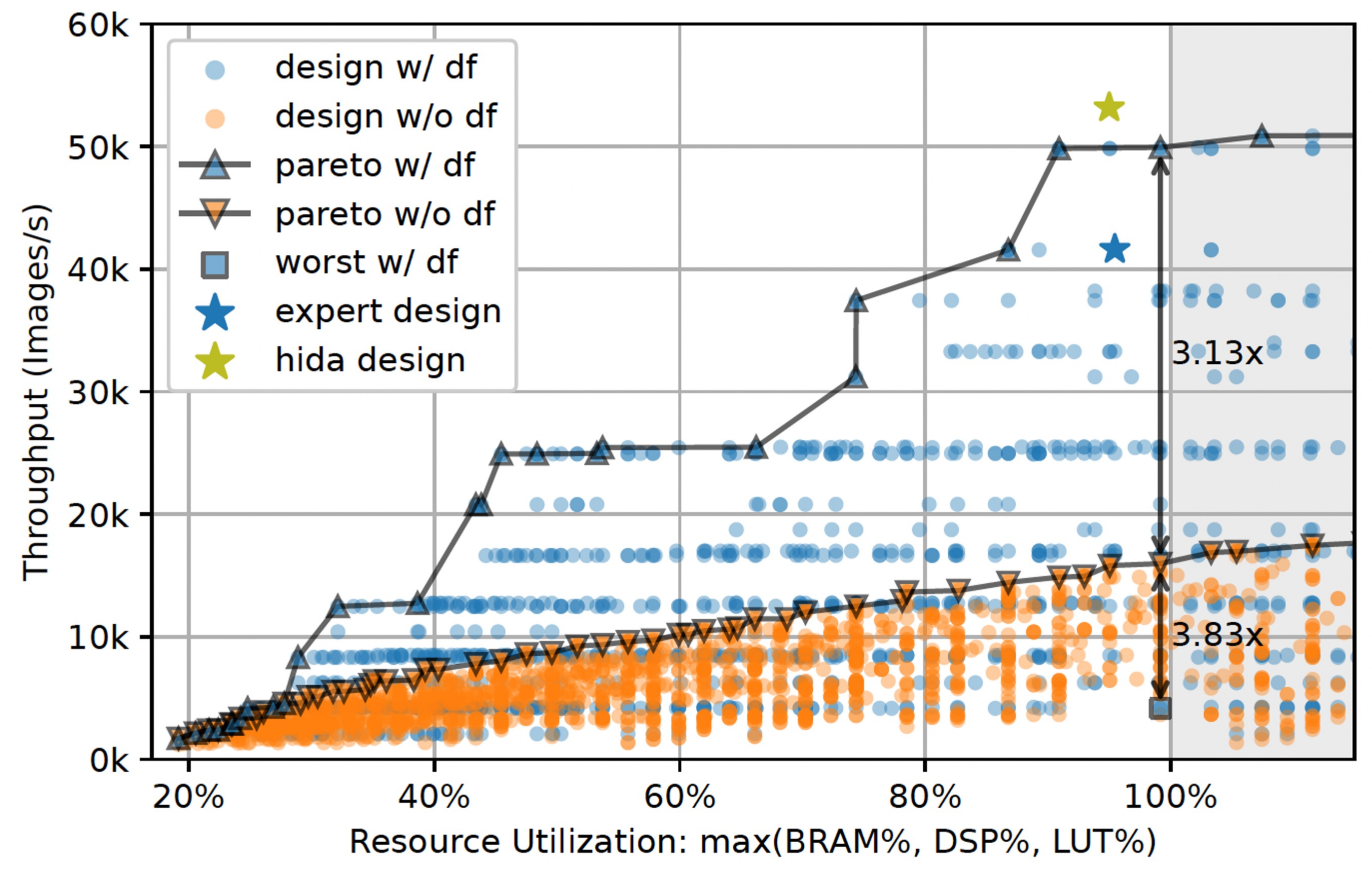


Non-dataflow Architecture



Dataflow Architecture

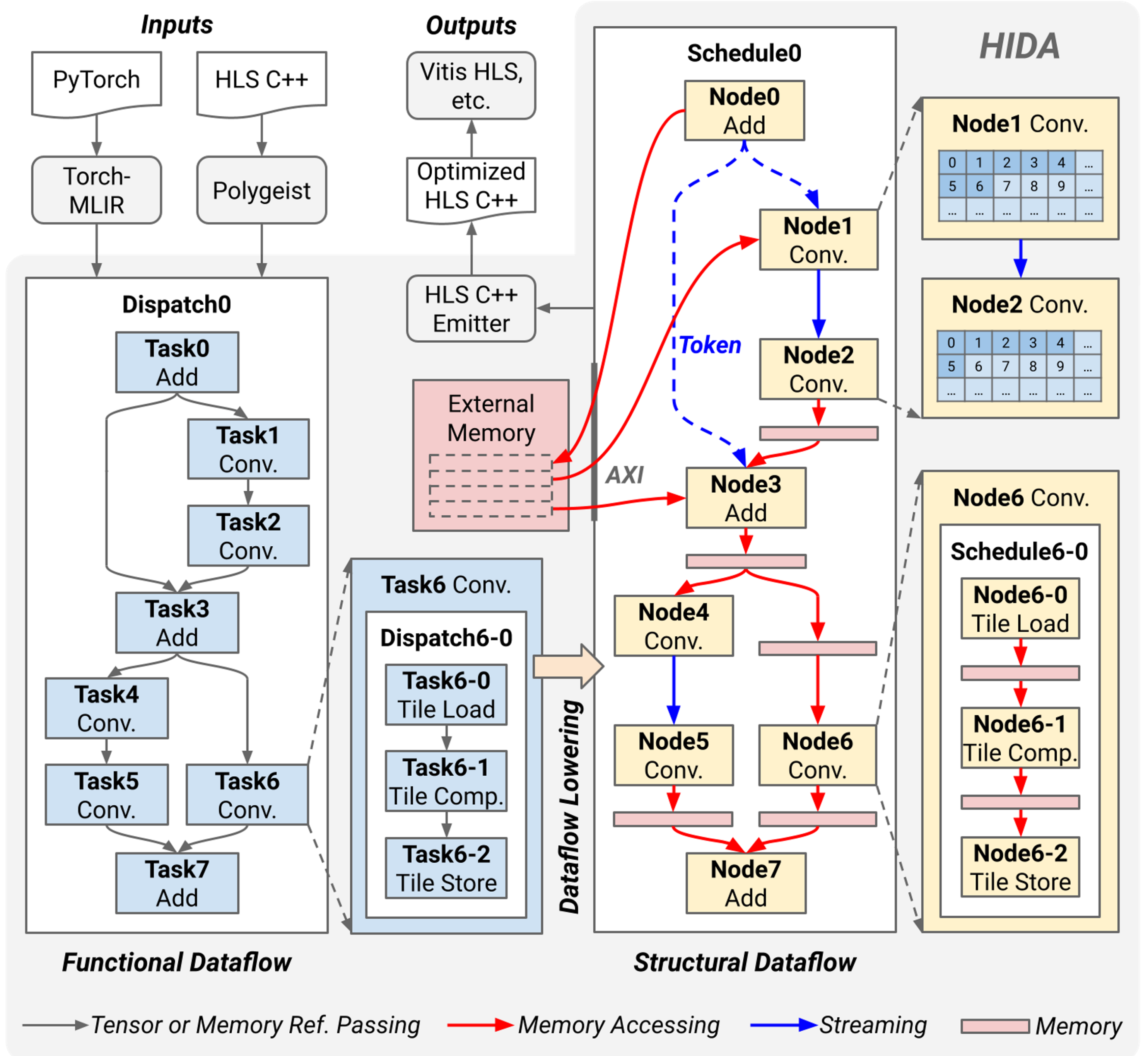
- **Keep intermediate data on chip** – reduce external memory access
- **Overlap task execution** – reduce on-chip memory utilization



Case Study: An LeNet Accelerator on FPGA

- Dataflow designs are Pareto dominating
- Dataflow designs cannot guarantee a good trade-off

HIDA Framework



HIDA Framework Overview

Functional Dataflow

```
%tensor = hida.task() : tensor<64x64xi8> { ... }
hida.task() { ... %tensor ... }
```

Structural Dataflow

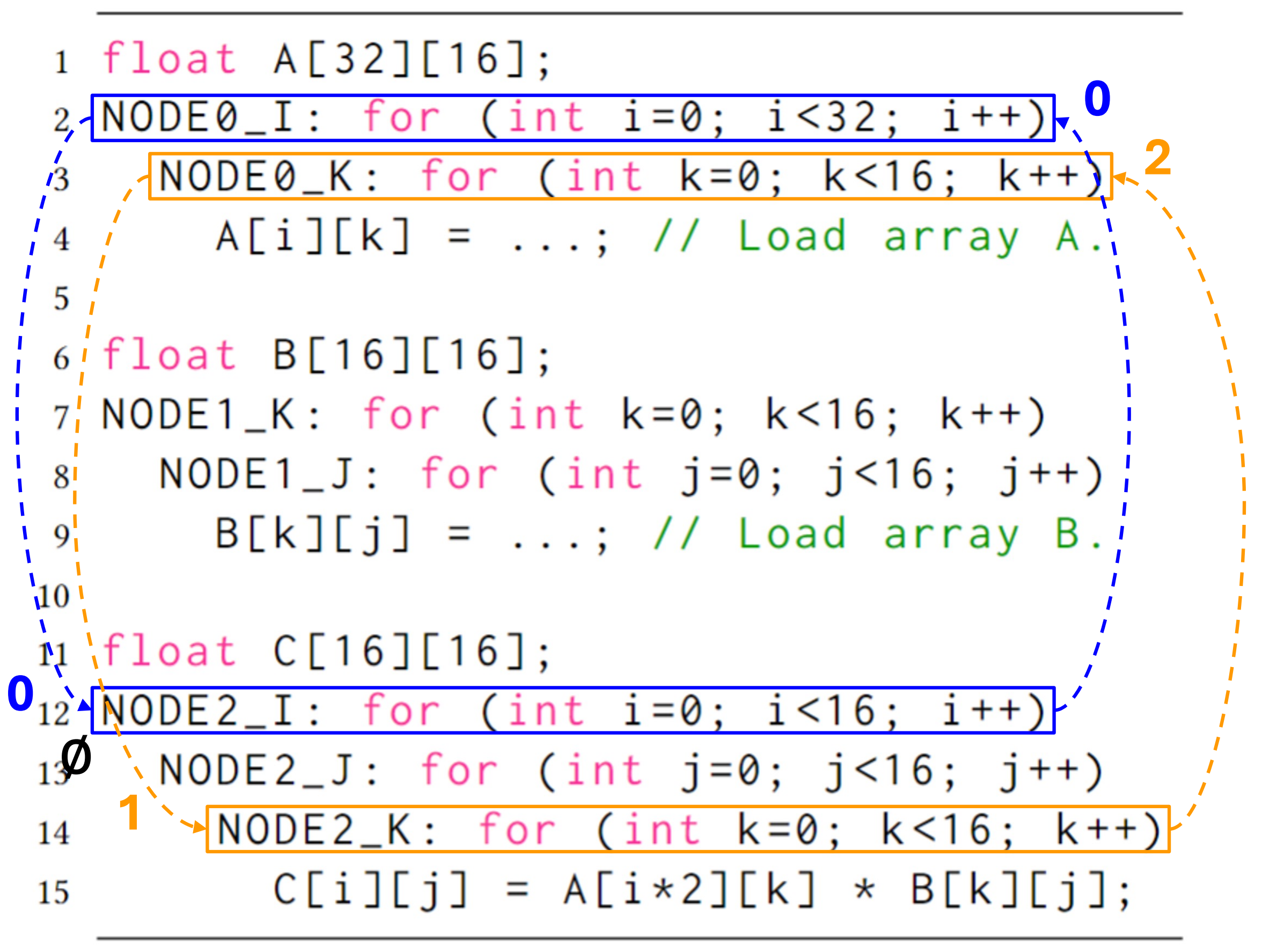
```
%buffer = hida.buffer : memref<64x64xi8, ...>
hida.node() -> (%buffer : memref<64x64xi8, ...>) { ... }
hida.node(%buffer : memref<64x64xi8, ...>) -> () { ... }
```

High-level Dataflow Optimizations ⟺ ***Low-level Dataflow Optimizations***

- Tensor & Linear algebra optimizations
  - Tiling, fusion, permutation, packing, etc.
- Full tensor reduction
  - Reducing full tensor to partial tensors
- Task manipulation
  - Placement, scheduling, etc.

- Ping-pong Buffer optimizations
  - Placement, partitioning, etc.
- Stream channel optimizations
  - Placement, vectorization, sizing, etc.
- Task optimizations
  - Pipelining, vectorization, etc.

HIDA Design Space Exploration



```
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];
```

Step (1) Connectedness Analysis

Source	Target	Buffer	Permutation Map		Scaling Map	
			S-to-T	T-to-S	S-to-T	T-to-S
Node0	Node2	A	[0, ∅, 1]	[0, 2]	[0.5, 1]	[2, ∅, 1]
Node1	Node2	B	[∅, 1, 0]	[2, 1]	[1, 1]	[∅, 1, 1]

- Permutation Map - Record the alignment between loops
- Scaling Map - Record the alignment between strides

Step (2) Node Sorting

Node	Connectedness	Intensity
Node0	1	512
Node1	1	256
Node2	2	4096

- Descending Order of Connectedness
- Computation Intensity as Tie-breaker

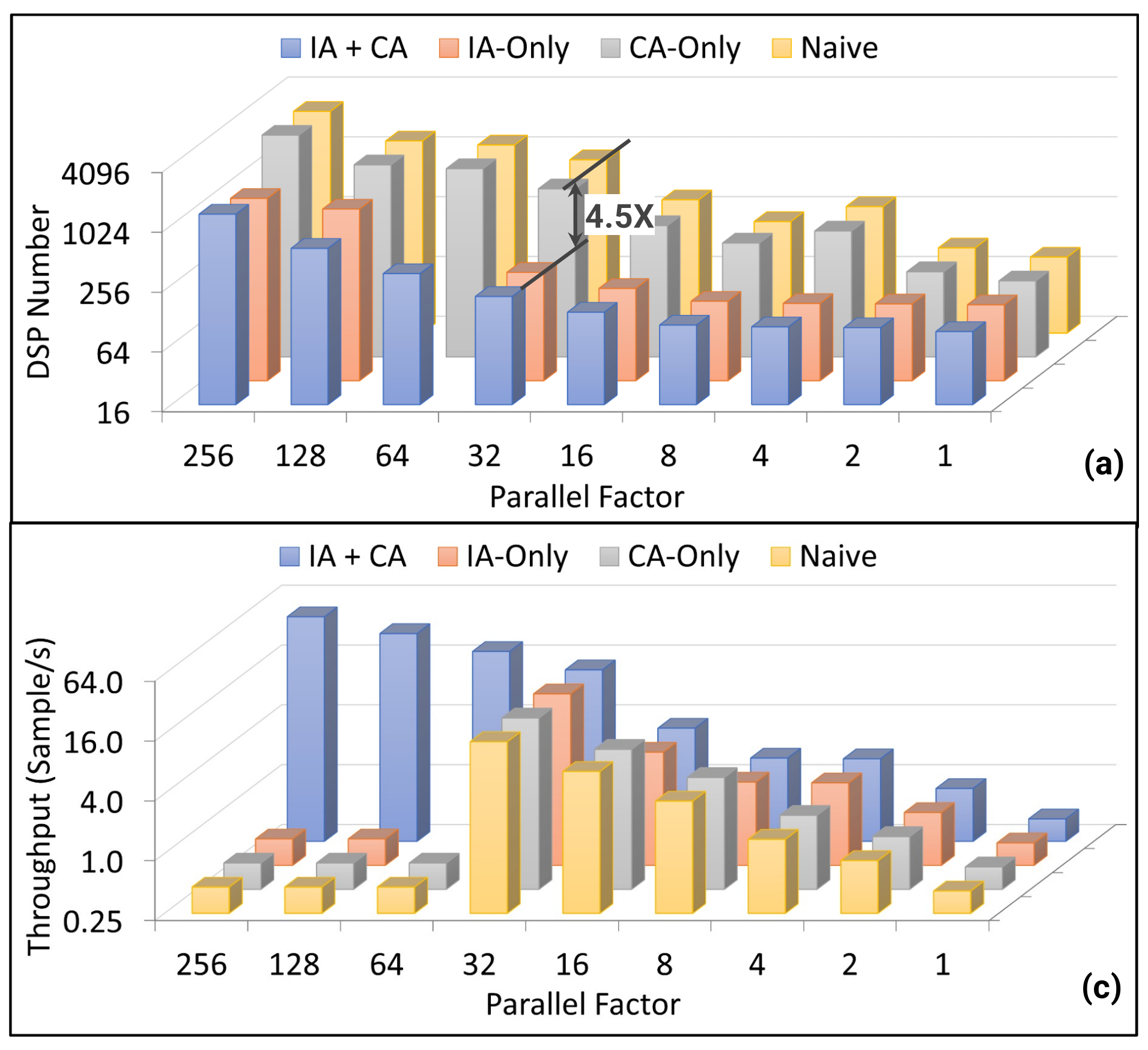
Step (3) Node Parallelization

- **Assuming maximum parallel factor is 32**
- **Node2 Parallelization: [4, 8, 1]**
  - Overall parallel factor is 32
  - Local DSE without constraints
  - Solution unroll factors: [4, 8, 1]
- **Node0 Parallelization: [4, 1]**
  - Overall parallel factor is 4, calculated from intensities of Node0 and 2 (32*512/4096)
  - Local DSE with connectedness constraints, the unroll factors must NOT be mutually indivisible with constraints
    - Multiply with scaling map:
    - [4, 8, 1] ⊙ [2, ∅, 1] = [8, ∅, 1]
    - Permute with permutation map:
    - permute([8, ∅, 1], [0, 2]) = [8, 1]
  - Solution unroll factors: [4, 1]

Experimental Results

Model	HIDA Compile Time (s)	Throughput (Samples/s)*			DSP Efficiency*		
		HIDA	DNNBuilder [75]	ScaleHLS [68]	HIDA	DNNBuilder [75]	ScaleHLS [68]
ResNet-18	83.1	45.4	-	3.3 (13.88×)	73.8%	-	5.2% (14.24×)
MobileNet	110.8	137.4	-	15.4 (8.90×)	75.5%	-	9.6% (7.88×)
ZFNet	116.2	90.4	112.2 (0.81×)	-	82.8%	79.7% (1.04×)	-
VGG-16	199.9	48.3	27.7 (1.74×)	6.9 (6.99×)	102.1%	96.2% (1.06×)	18.6% (5.49×)
YOLO	188.2	33.7	22.1 (1.52×)	-	94.3%	86.0% (1.10×)	-
MLP	40.9	938.9	-	152.6 (6.15×)	90.0%	-	17.6% (5.10×)
Geo. Mean	108.7		1.29×	8.54×		1.07×	7.49×

HIDA Results on DNN Models



Open-sourced on GitHub