Parashar ISCA ‘17

SCNN: An Accelerator for Compressed-sparse Convolutional Neural Networks

Convolutional Neural Nets

- Series of layers, generate output activations (OA) becoming input activations (IA) to next layer
- Convolution (1x1, 3x3, 5x5 filters with trained weights)
- Non-linear scalar operator (e.g., ReLU, clamping negatives to 0)
- Downsample
- Can use separate systems for training and inference (focus on latter)

Seven Dimensions

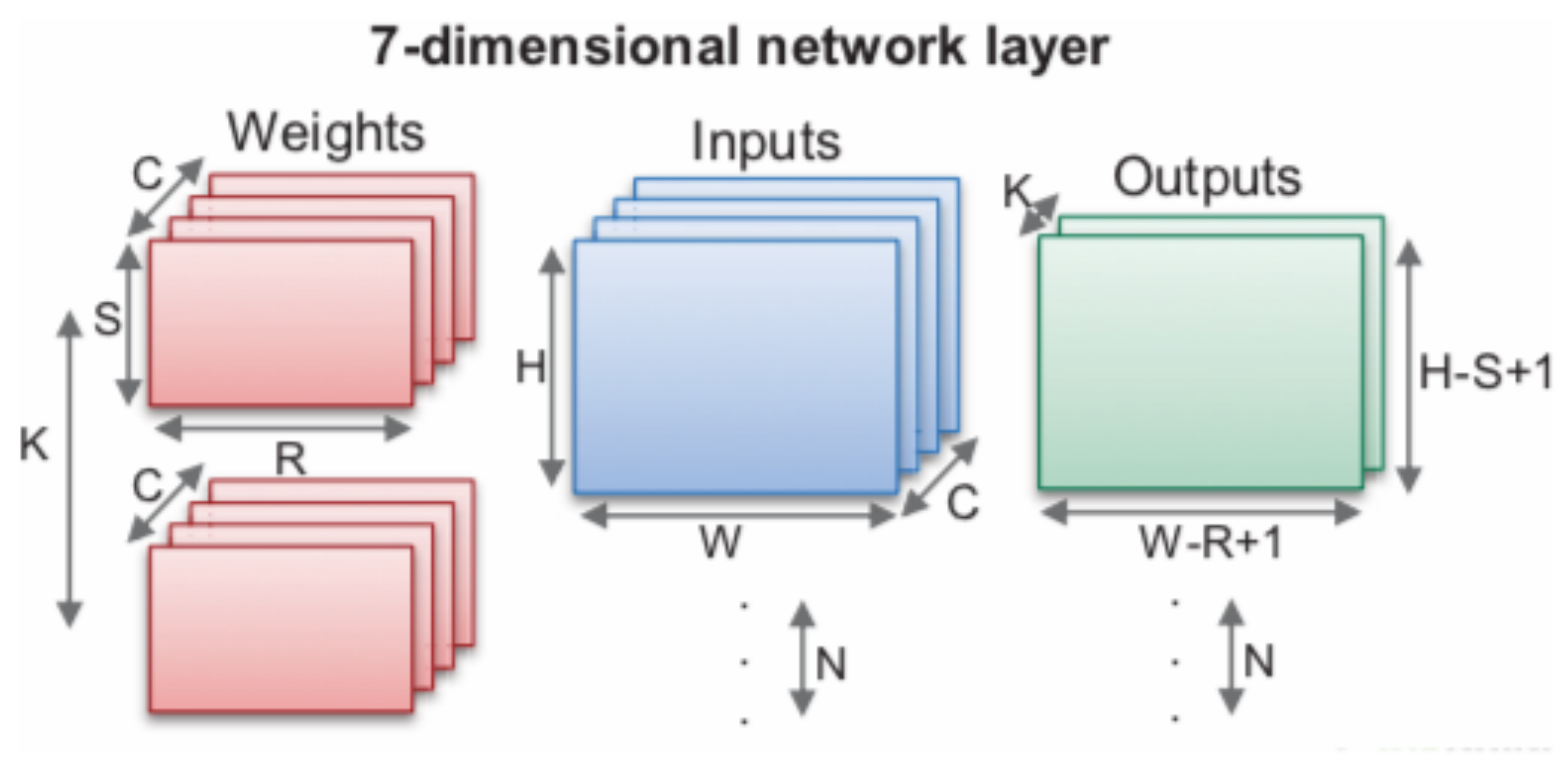


Figure 2: CNN computations and parameters.

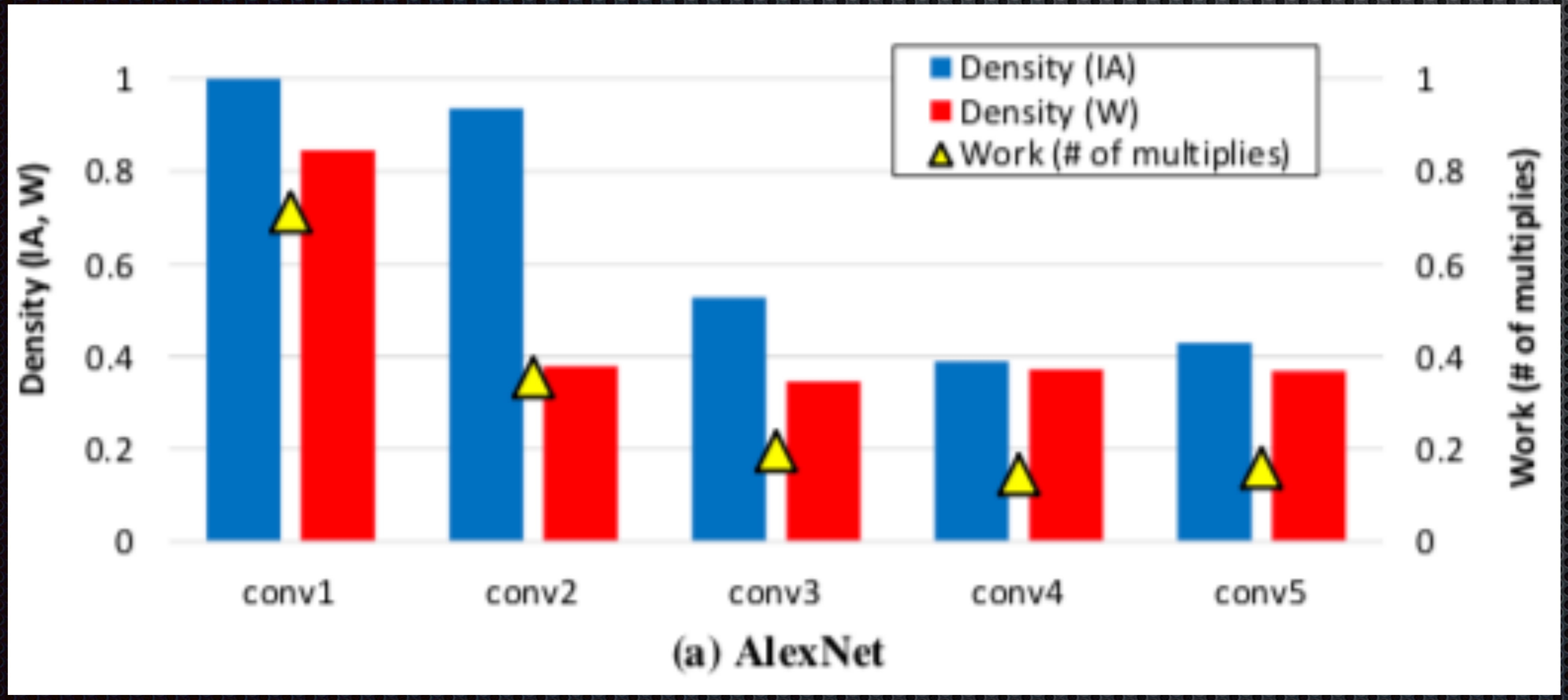
```
for n = 1 to N
  for k = 1 to K
    for c = 1 to C
      for w = 1 to W
        for h = 1 to H
          for r = 1 to R
            for s = 1 to S
              out[n][k][w][h] +=
                in[n][c][w+r-1][h+s-1] *
                filter[k][c][r][s];
```

Figure 3: 7-dimensional CNN loop nest.

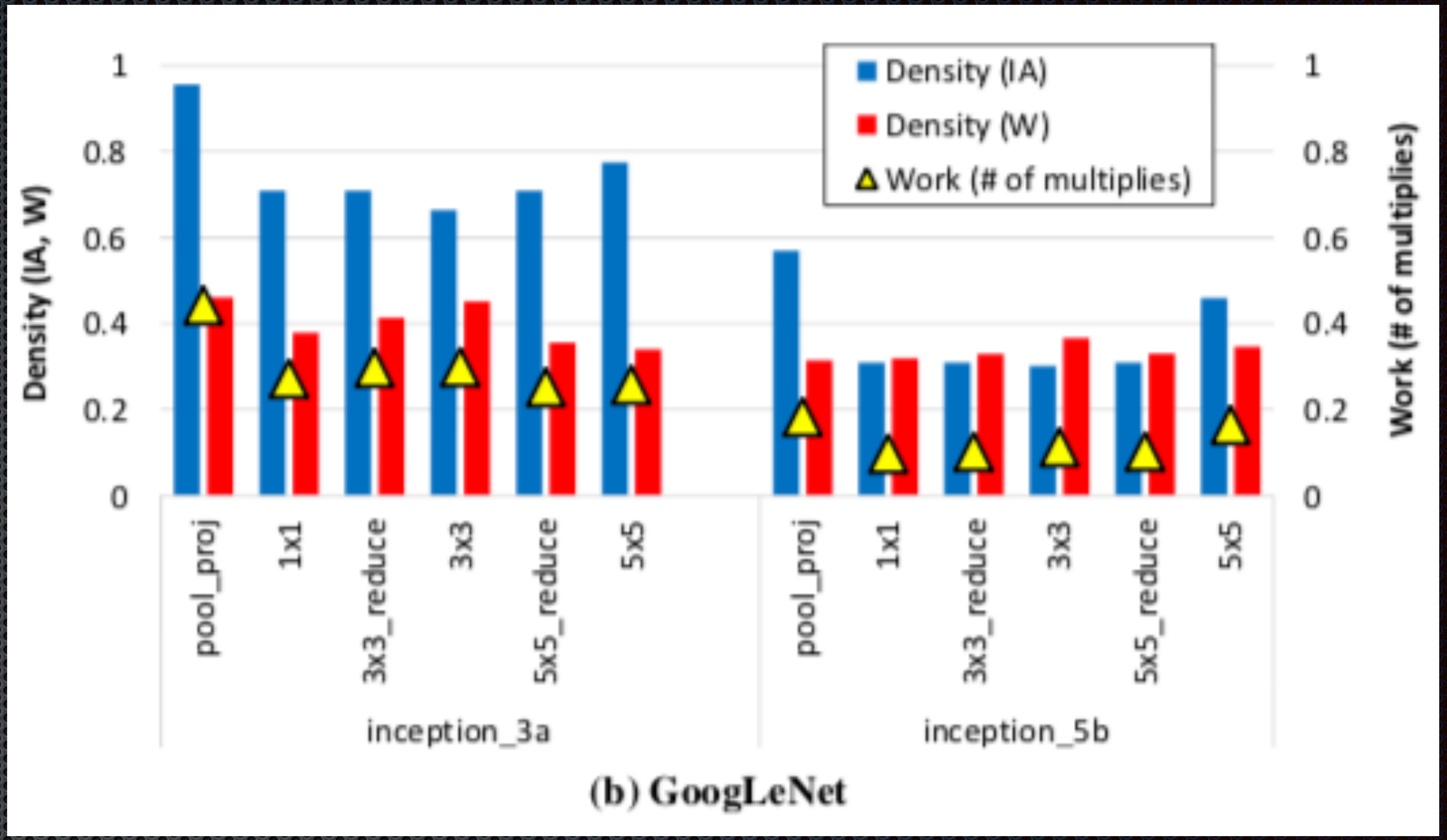
Plenty of opportunity for parallelism

Sparsity

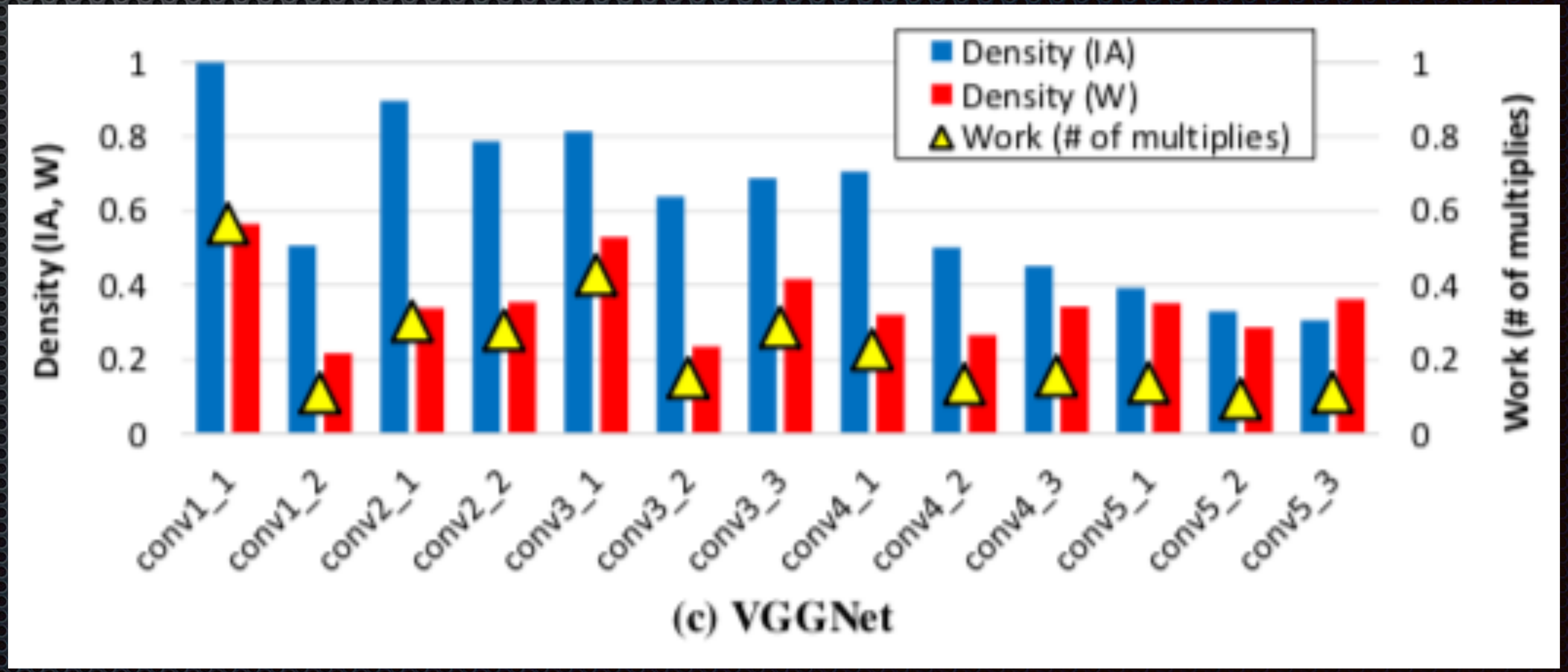
- Zeros in a layer's weight and input activation matrices — generate 0s out
- ReLU produces many zeros



(a) AlexNet



(b) GoogLeNet



(c) VGGNet

Exploiting Sparsity

- Compressing data
  - Reduces data movement, and thus saves energy
  - More efficient use of memory, allowing larger networks
- Reducing computation
  - Less time (zeros don't have to be multiplied, fewer multipliers)
  - Saves energy for fixed problem size or enables larger problems
- Energy is important for inference in deployed mobile applications

General Dataflow

- Different ways to get the computations done, which data has to move
- Input stationary (IS) keeps the input activation (C) plane fixed in memory
- K filters are applied over inputs to give K output channels. N groups of input channels can be passed to the filters
- Results in an N -> K -> C loop nest
- Within that nest is the WxH element output, followed by the RxS reduction and scaling filters

PT-IS-CP Dataflow

- Break output channels into blocks that can be reused (K/K_C)
- Weights buffer volume: $C \times K_C \times R \times S$
- Inputs buffer volume: $C \times W \times H$
- Partial sums volume: $K_C \times W \times H$
- Results in reordered loop nest: K/K_C -> C -> W -> H -> K_C -> R -> S

Cartesian Product (CP)

- Vector of F filter weights multiplied by vector of I inputs
- F weights are multicast to all I activations
- With compressed representation, all multiplies will be useful
- Accumulation unit as F x I adders

Planar Tiles

- Scaling to multiple processors
- Divide each layer into tiles, W_t x H_t
- All of input activation layer C goes to each tile
- Outputs generate halos that extend beyond tiles
  - Halos get transmitted to adjacent processors for summing

Processor Arrangement

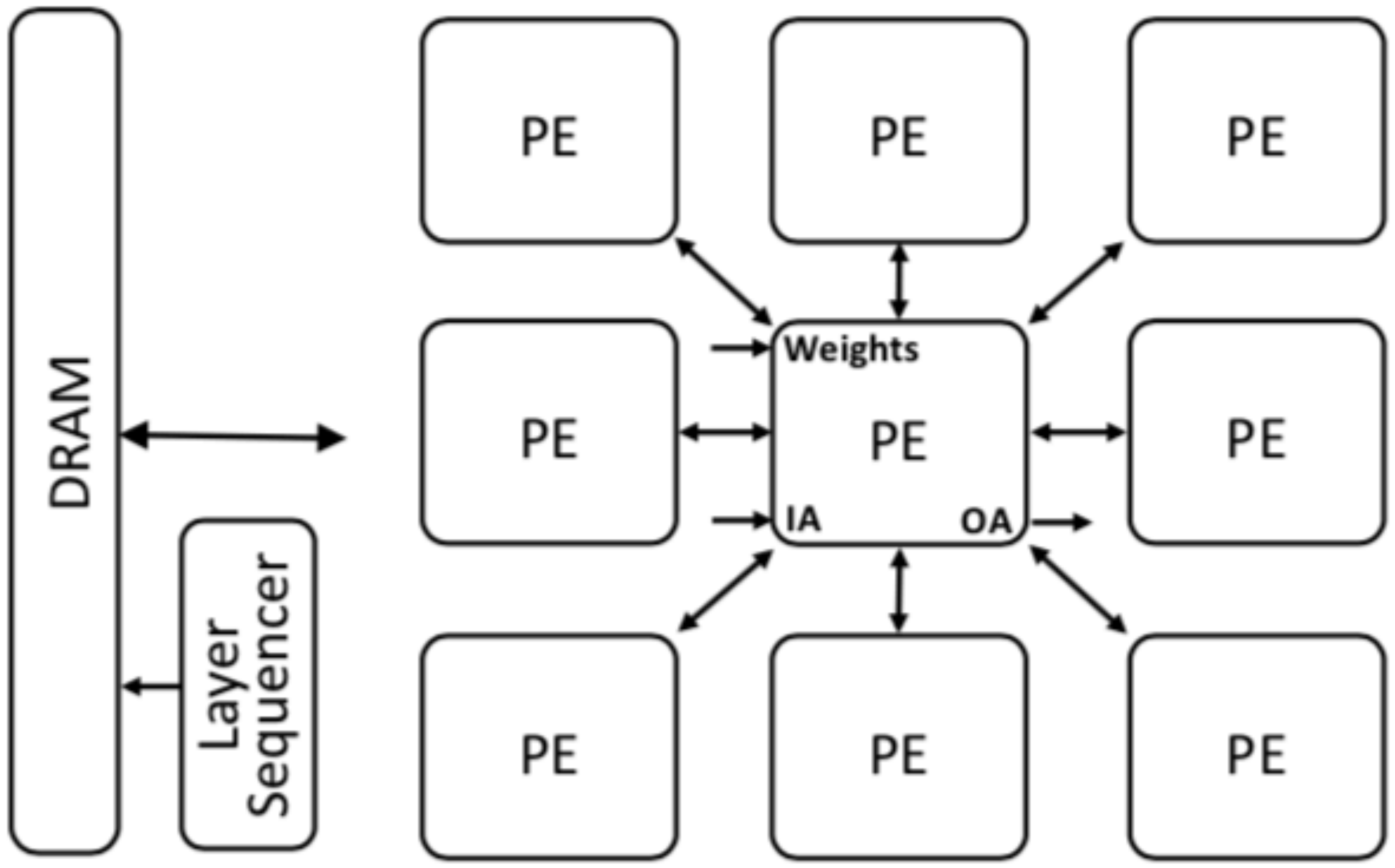


Figure 5: Complete SCNN architecture.

Compression Representation

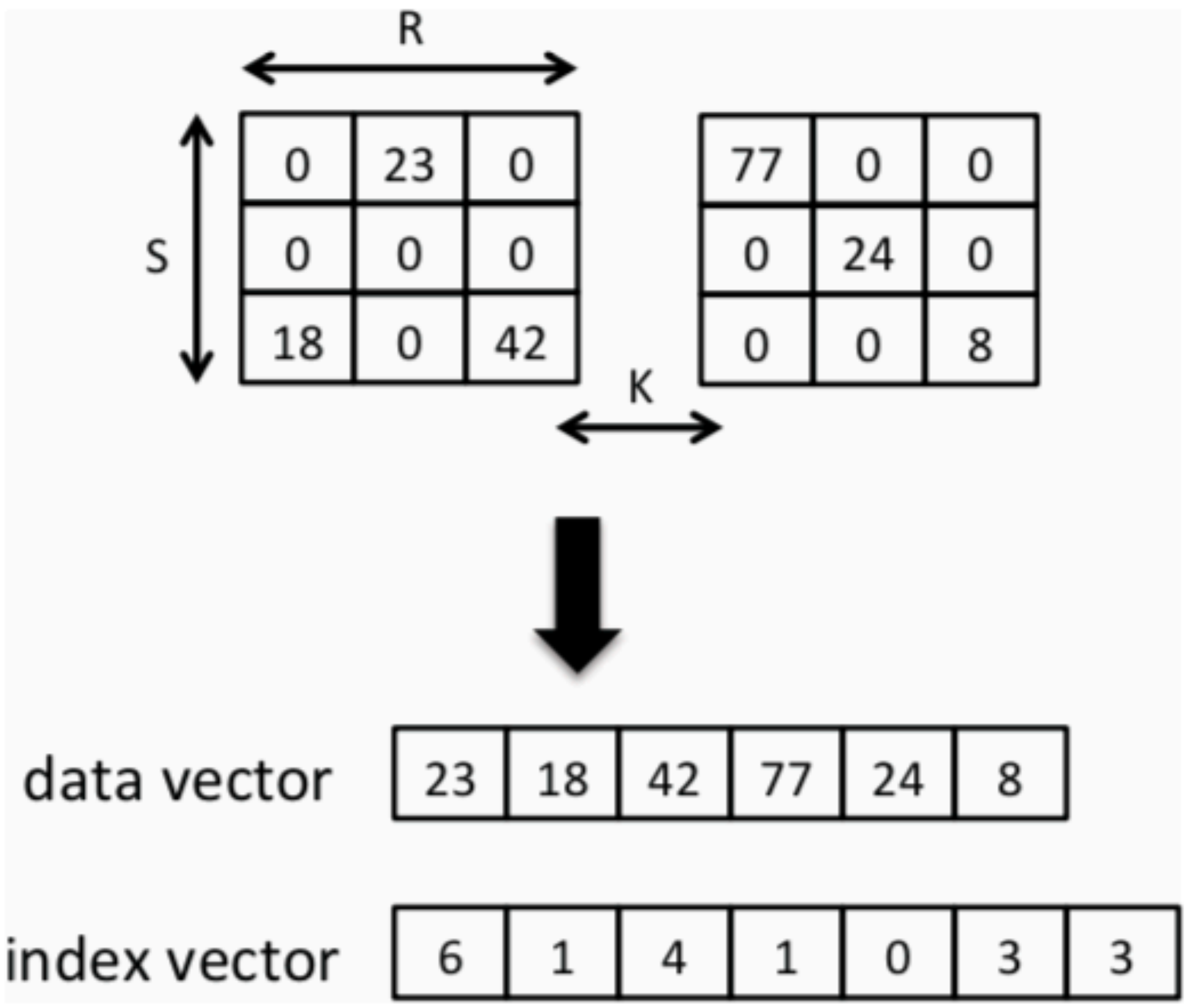


Figure 7: Weight compression.

Index vector is #non-0s, then #0s before each data vector entry

Processor Architecture

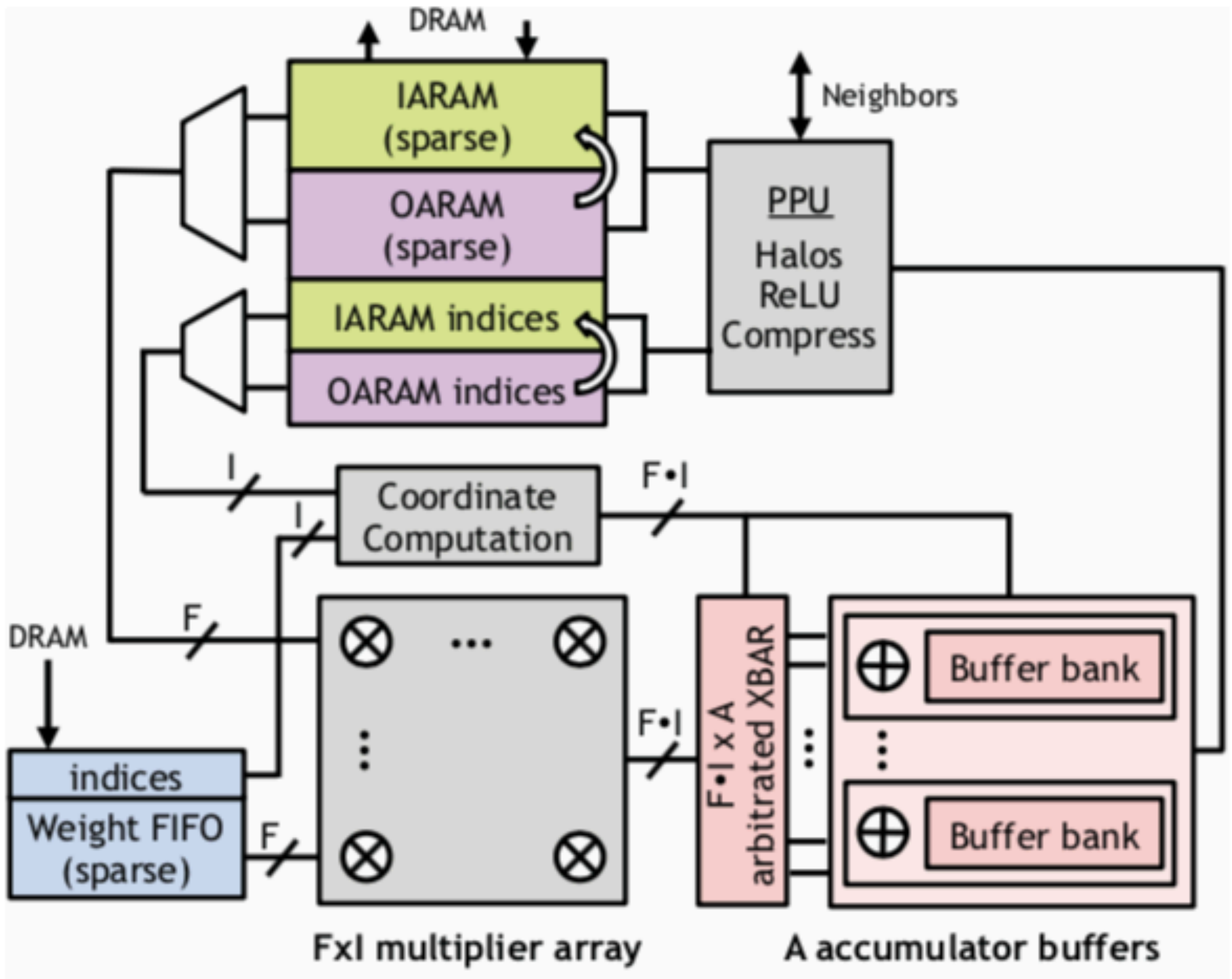


Figure 6: SCNN PE employing the PT-IS-CP-sparse dataflow.

Where not to compress

- The adders need to generate spatially mapped outputs
- The multiplier outputs are sent via index array to adders so that outputs at the same location can be summed
- Adders can be aliased, so there are extra (2x) to reduce that
- Once the sums are produced, they are reduced and scaled and can then be compressed again

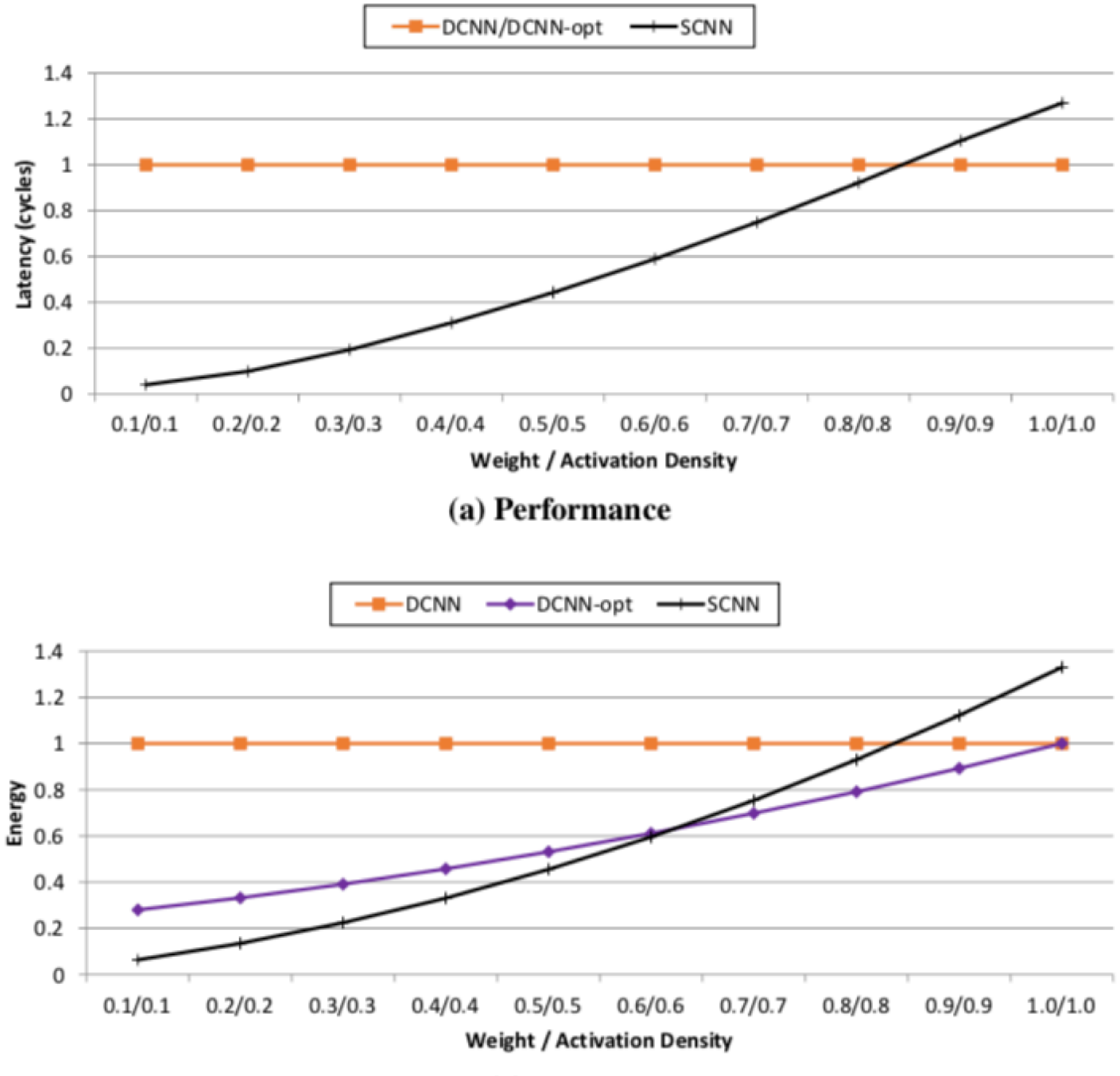
Issues

- Fully connected layers are hard to handle with compression, but are rare
- Large models may not fit, and have to be swapped out to RAM (called temporal tiling)
  - CNN designers may rework their nets to fit for mobile applications

Overall Parameters

- 64 processors, total of 7.9 mm^2
- 1024 multipliers (16 per processor)
- 2MB of SRAM
- Developed with CAD tools to full layout and circuit simulation
- Validated cycle-level simulator and separate analytical tool
- Also designed dense version, and energy-optimized dense version for comparison, and optimal (oracle) version

Sparsity Sensitivity Analysis



(a) Performance

(b) Energy

Figure 8: GoogLeNet performance and energy versus density.

Performance

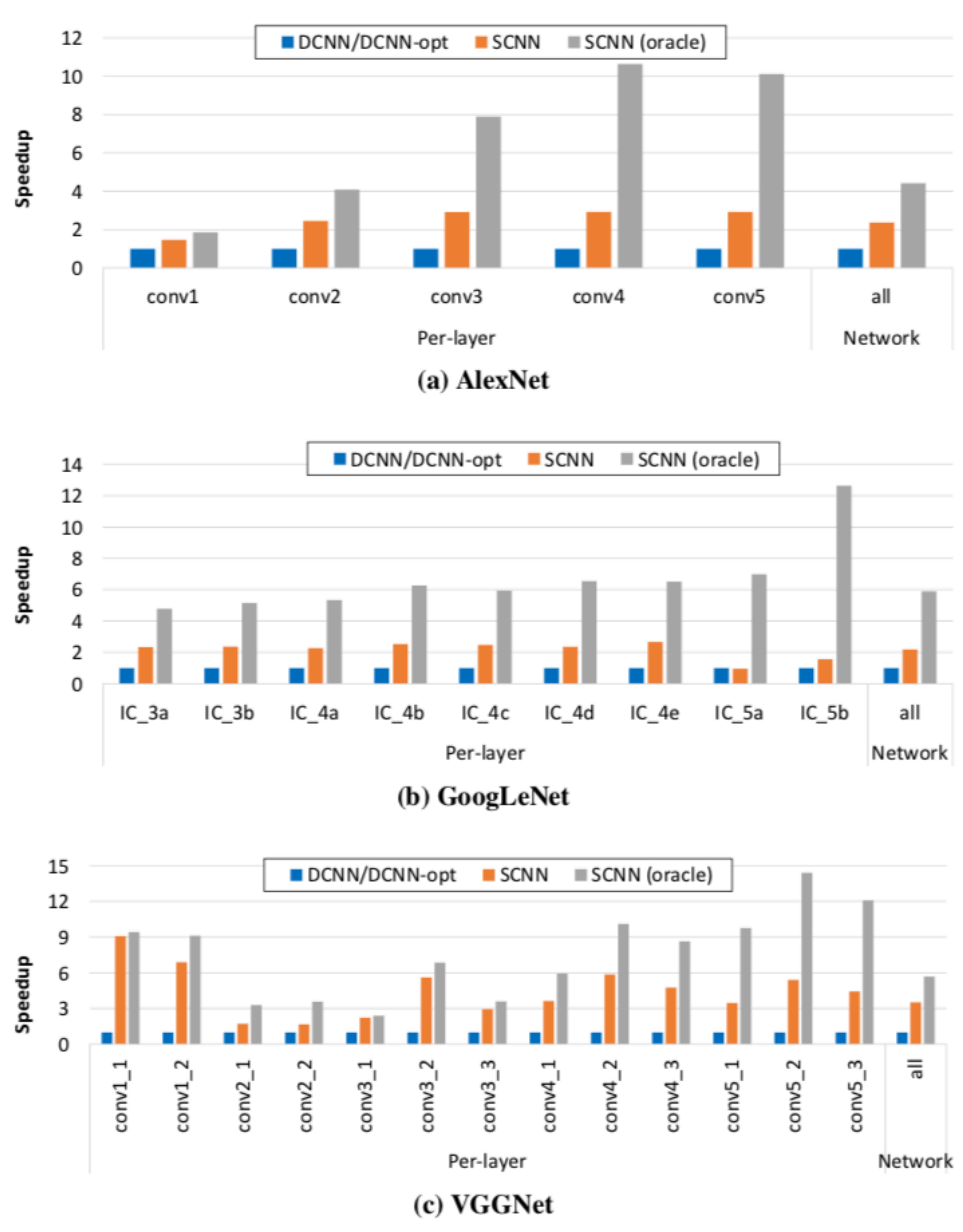


Figure 9: SCNN performance comparison.

Multiplier Utilization

- Later layers can have small working sets, so multipliers go idle
- Tiles can have unbalanced loads, so some wait for others to finish

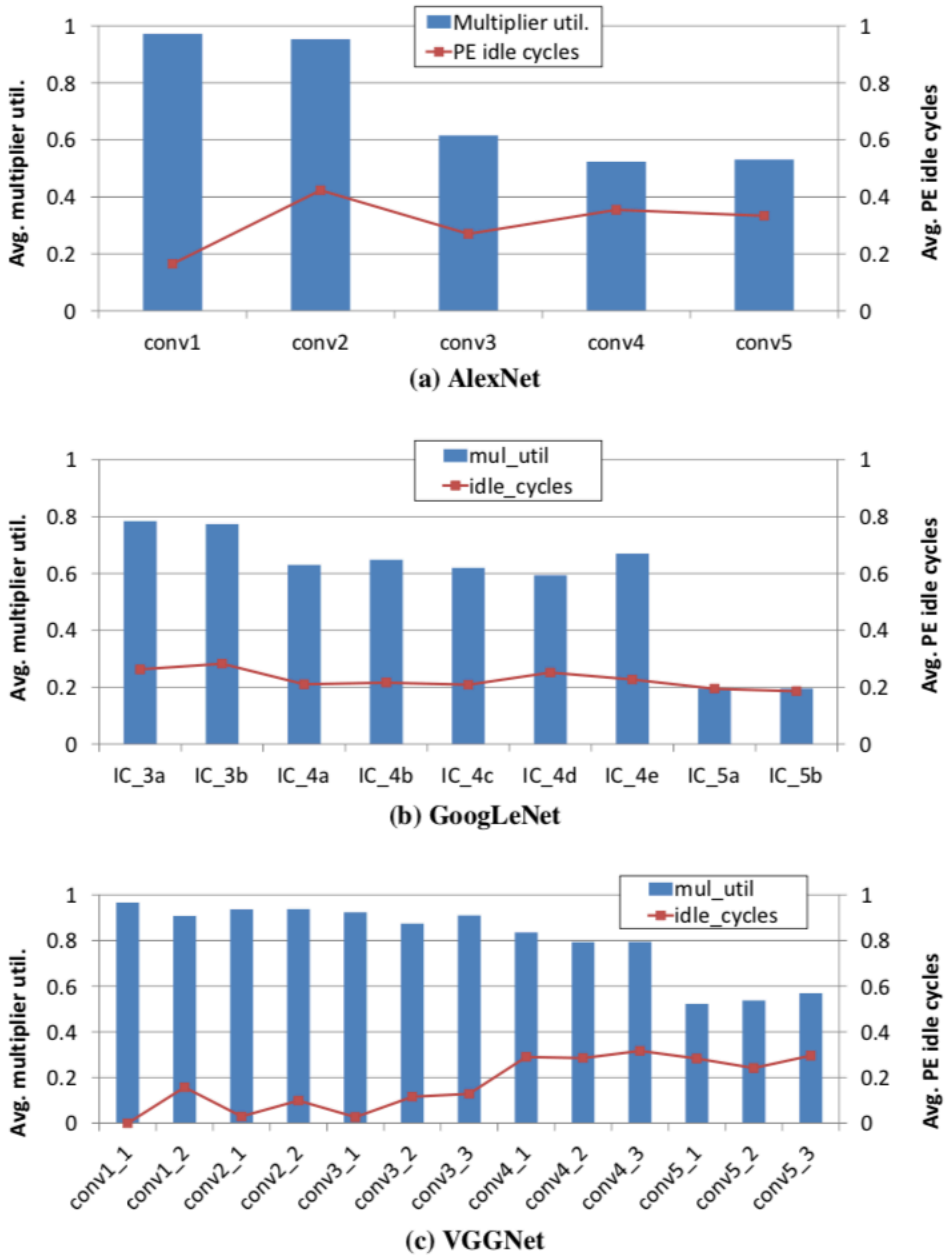


Figure 10: Average multiplier array utilization (left-axis) and the average fraction of time PEs are stalled on a global barrier (right-axis), set at the boundaries of output channel groups.

Energy Efficiency

Surprise that DNN-Opt does so well

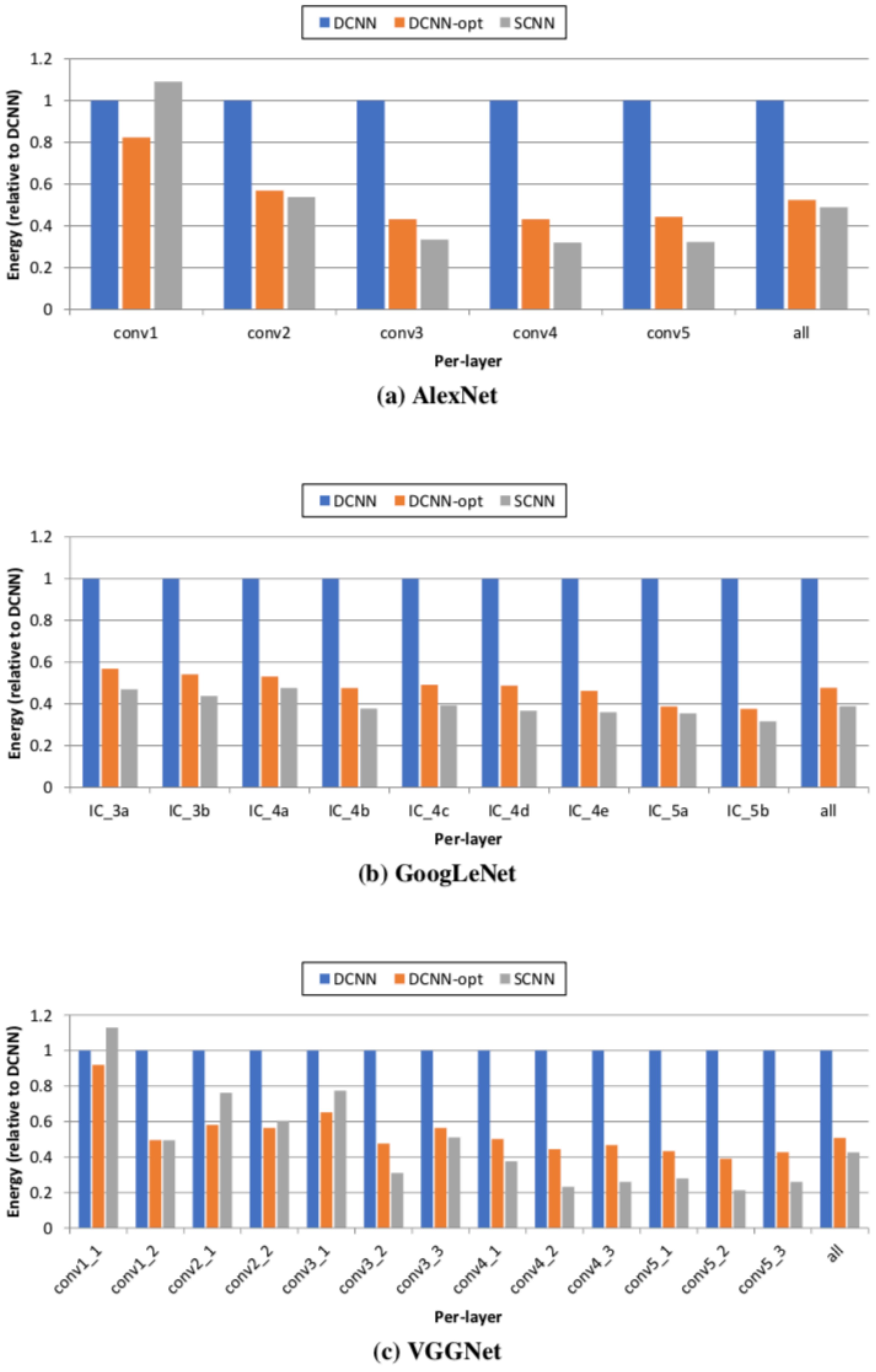


Figure 11: SCNN energy-efficiency comparison.

Effect of Density

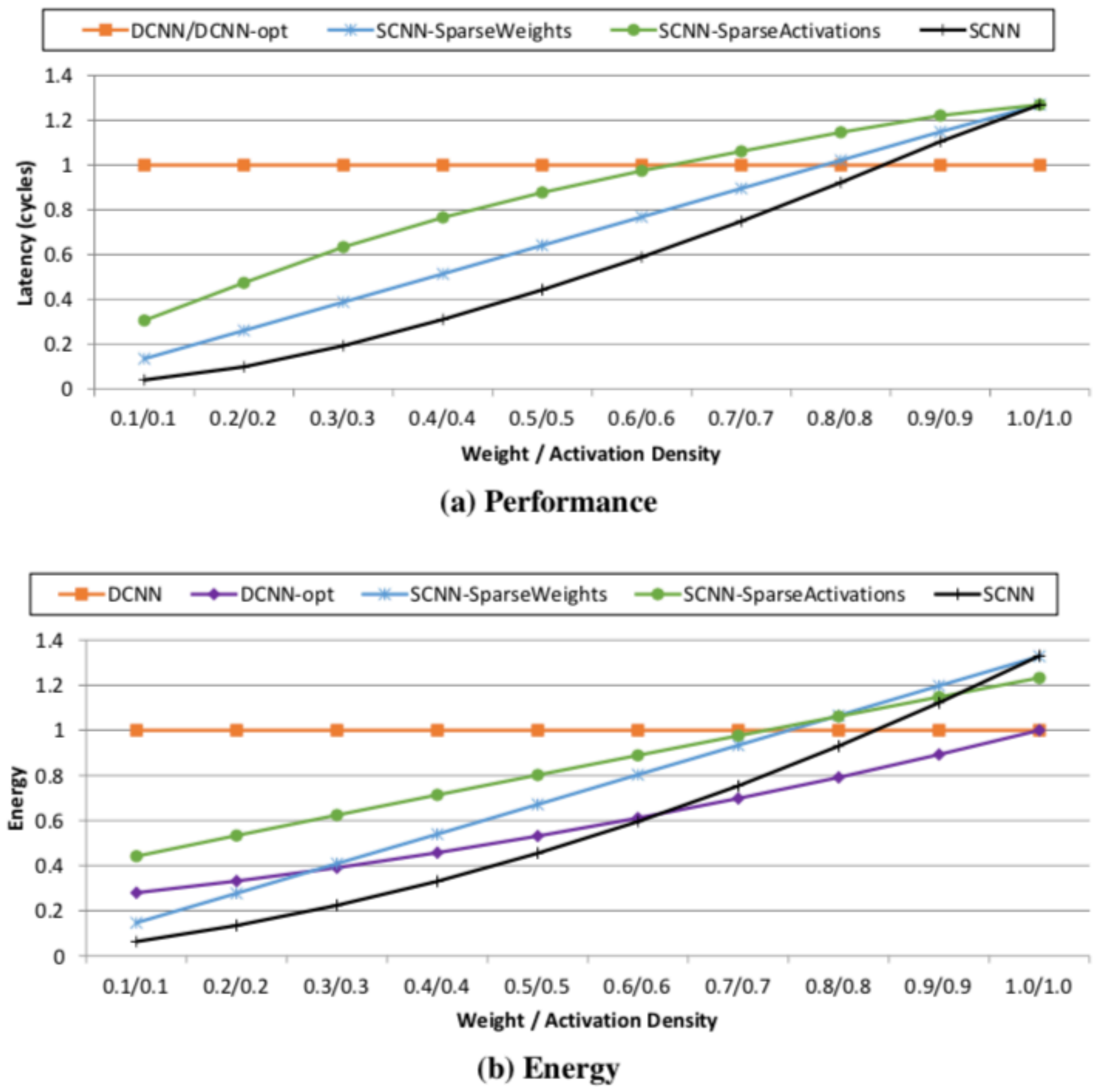


Figure 12: GoogLeNet performance and energy versus density for sparse weights, sparse activations, and both

Discussion

Fowers ISCA18

A Configurable Cloud-Scale DNN Processor for Real-Time AI

Microsoft’s FPGA-based NN Processor

- Single or small batch sizes for inference on individual transactions
- Low latency - user-facing applications
- Network resident as a resource in a datacenter
- SIMD with extensive pipelining
- Co-processor, but with large granularity instructions for millions of ops

Placement in a Datacenter

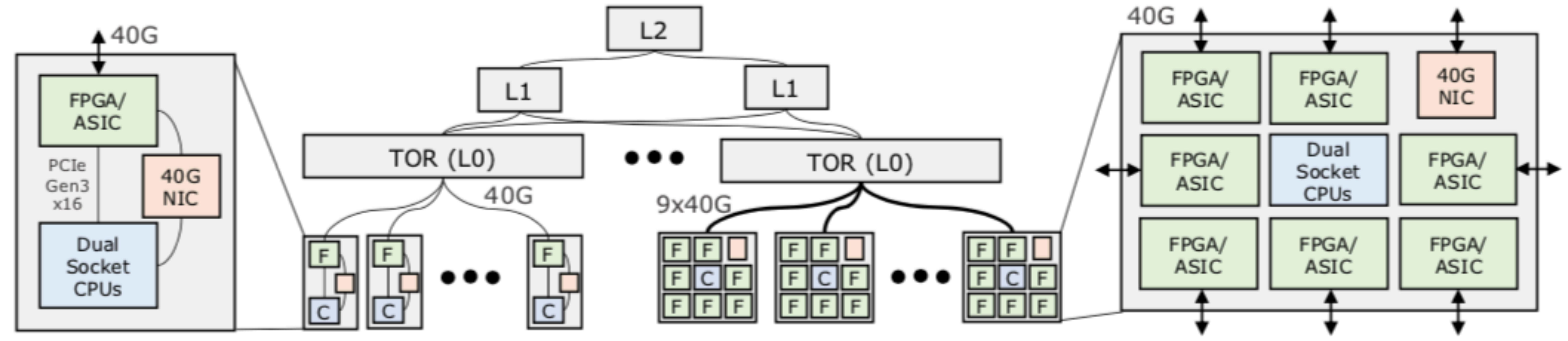


Fig. 1. BW system at cloud scale. From left to right, servers with bump-in-the-wire accelerators, accelerators connected directly to the hyperscale datacenter network, an accelerator appliance.

LSTM Analysis

	Operations	Latency
1	$8N^2$ multiplies 8N N/2 add-reductions	$4N^2$ / $\#FU_{MVM}$* + log(N)
2	4N adds	4N / $\#FU_{Add}$
3	4N adds	4N / $\#FU_{Add}$
4	3N sigmoids N tanhs	3N / $\#FU_{Sigmoid}$ N / $\#FU_{Tanh}$
5	2N multiplies	2N / $\#FU_{Hadamard}$
6	N adds	N / $\#FU_{Add}$
7	N tanhs	N / $\#FU_{Tanh}$
8	N multiplies	N / $\#FU_{Hadamard}$

* FU_{MVM} = 1 multiply and 1 add

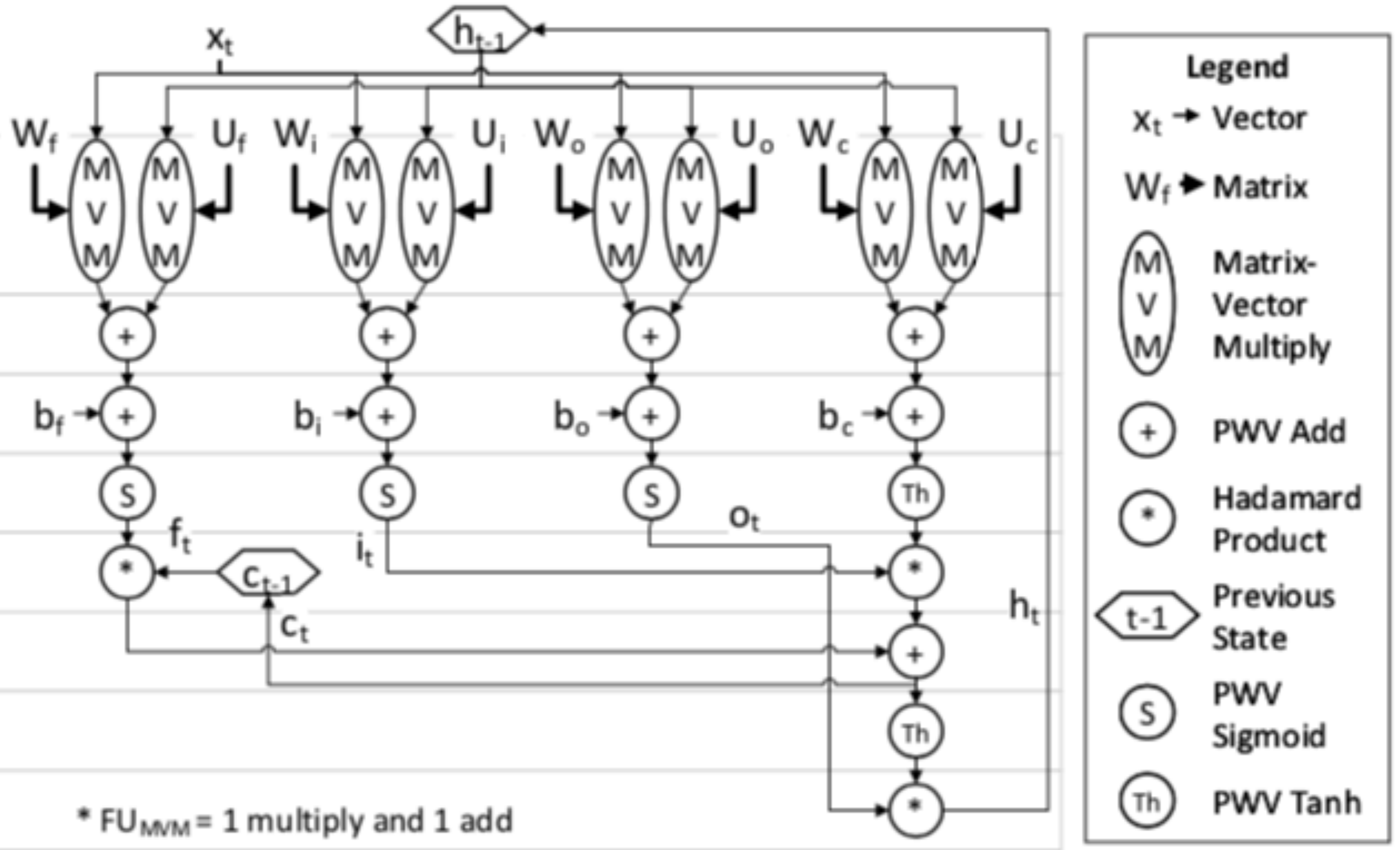


Fig. 2. LSTM critical-path analysis. Operation count and latency are shown as functions of LSTM dimension (N) and number of functional units (#FU).

This kind of analysis motivates the design to use an explicit chaining instruction set that sets up a dataflow-like vector path

Instruction Set

Name	Description	IN	Operand 1	Operand 2	OUT
v_rd	Vector read	-	MemID	Memory index	V
v_wr	Vector write	V	MemID	Memory index	-
m_rd	Matrix read	-	MemID (NetQ or DRAM only)	Memory index	M
m_wr	Matrix write	M	MemID (MatrixRf or DRAM only)	Memory index	-
mv_mul	Matrix-vector multiply	V	MatrixRf index	-	V
vv_add	PWV addition	V	AddSubVrf index	-	V
vv_a_sub_b	PWV subtraction, IN is minuend	V	AddSubVrf index	-	V
vv_b_sub_a	PWV subtraction, IN is subtrahend	V	AddSubVrf index	-	V
vv_max	PWV max	V	AddSubVrf index	-	V
vv_mul	Hadamard product	V	MultiplyVrf index	-	V
v_relu	PWV ReLU	V	-	-	V
v_sigm	PWV sigmoid	V	-	-	V
v_tanh	PWV hyperbolic tangent	V	-	-	V
s_wr	Write scalar control register	-	Scalar reg index	Scalar value	-
end_chain	End instruction chain	-	-	-	-

PWV = point-wise vector operation. IN = implicit input (V: vector, M: matrix, -: none). OUT = implicit output.

TABLE II
THE SINGLE-THREADED BW NPU ISA EXPOSES A COMPACT AND SIMPLE ABSTRACTION FOR TARGETING DNN MODELS.

A chain begins with a read input and ends with a write output

Architecture

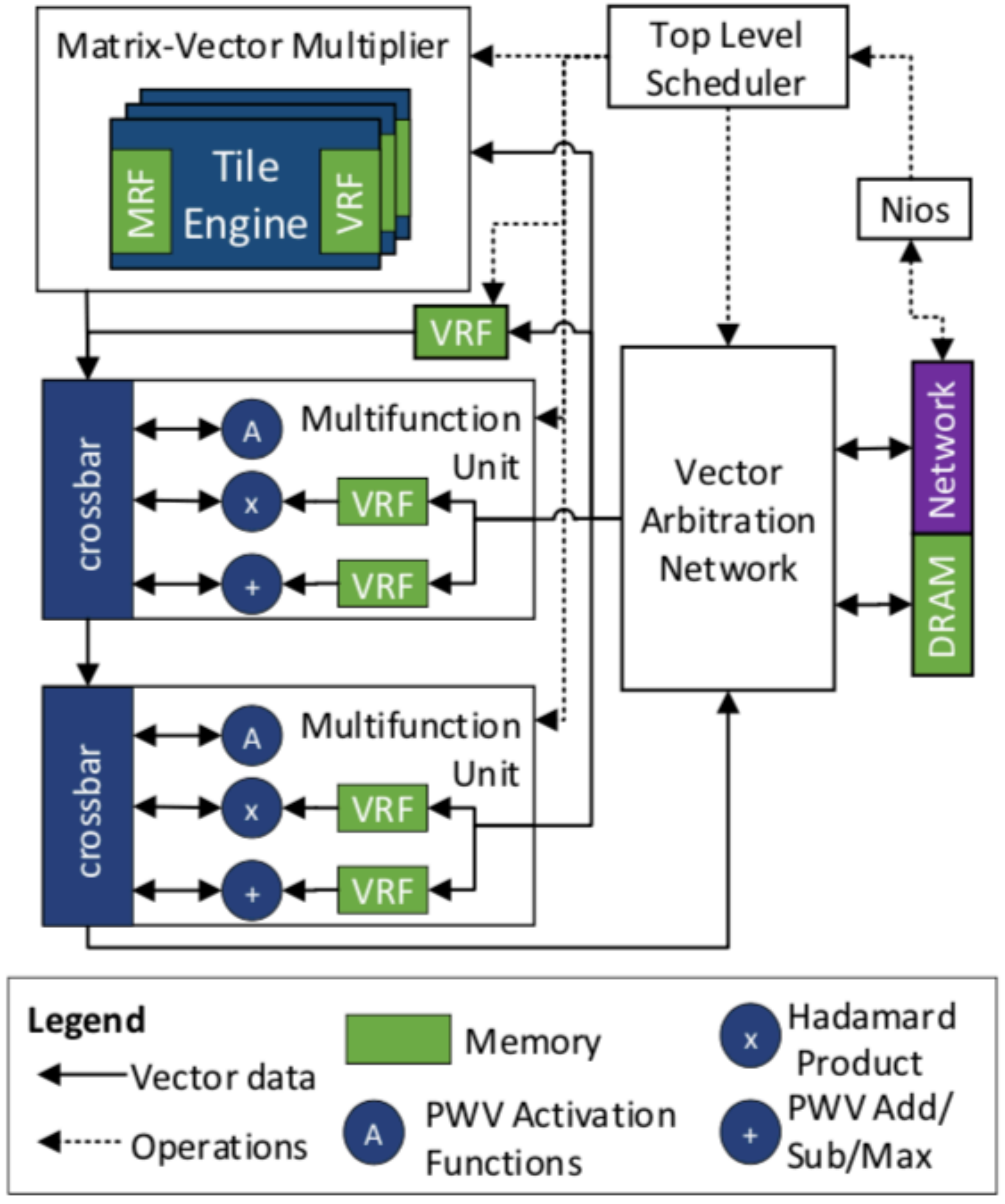


Fig. 3. Microarchitecture overview.

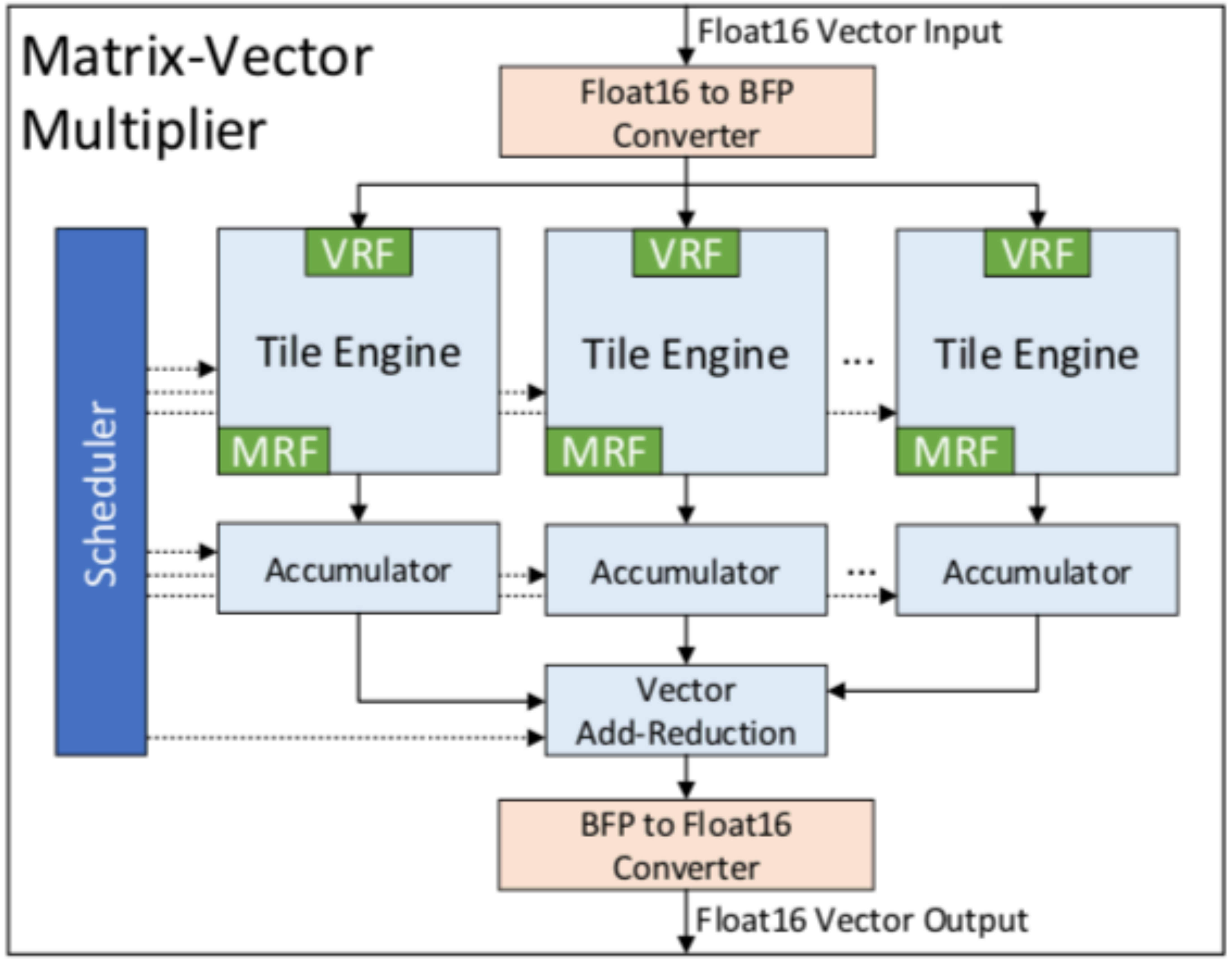


Fig. 4. Matrix-vector multiplier overview.

The SIMD Part

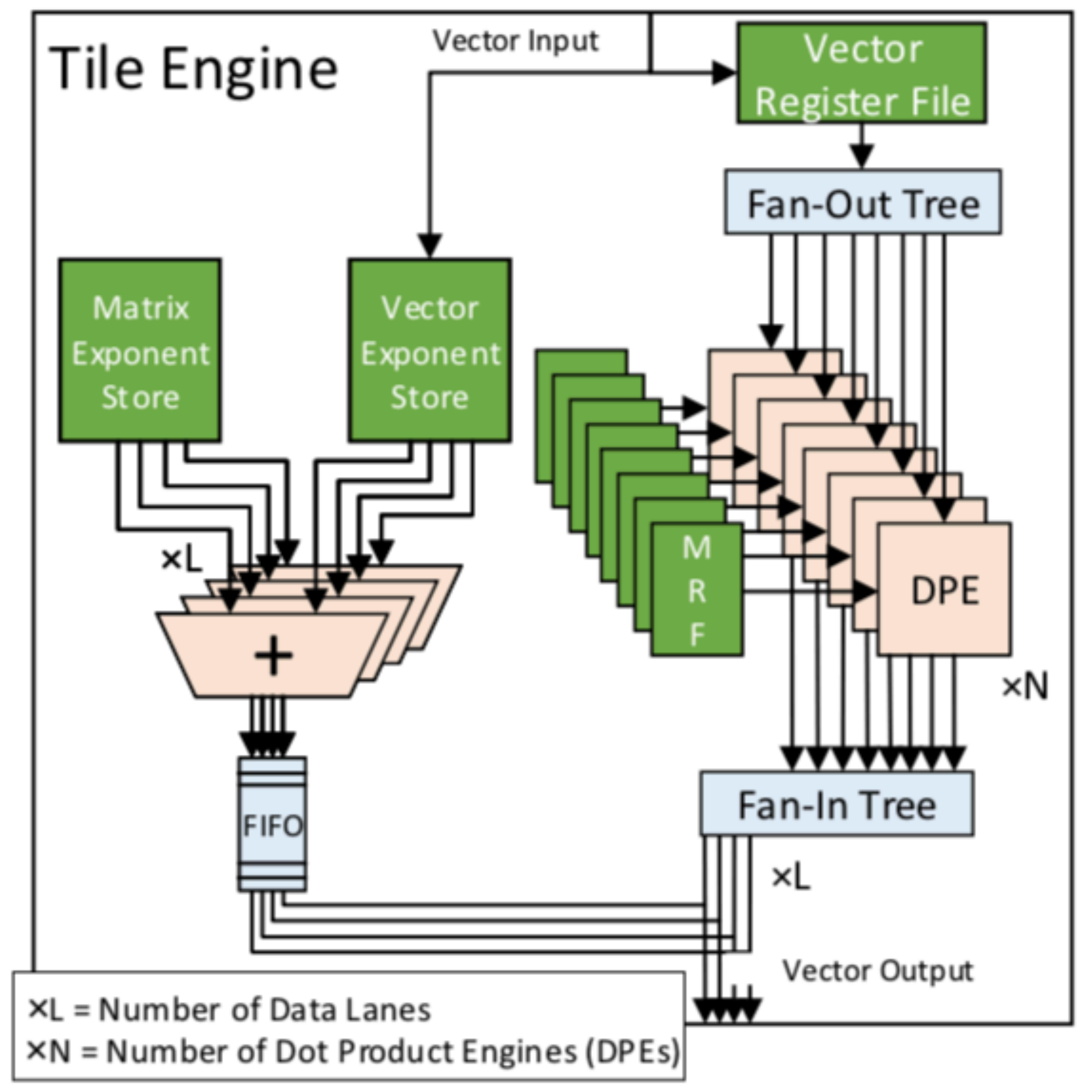


Fig. 5. Matrix-vector tile engine microarchitecture.

SIMD Issue is Hard

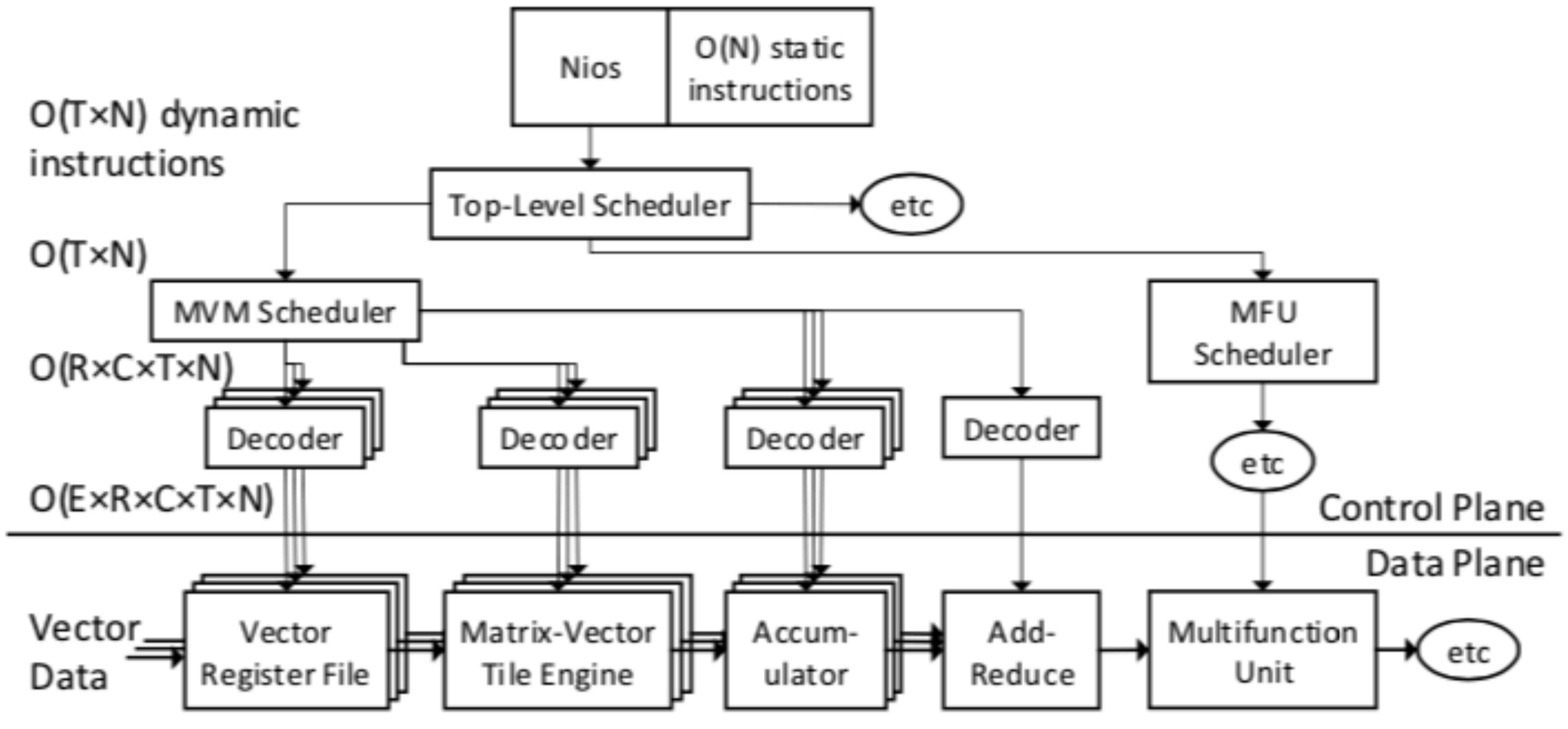


Fig. 6. Hierarchical decode and dispatch (HDD) into the matrix-vector multiplier.

Uses a hierarchy of schedulers and decoders to distribute control

Performance Compared to GPU

	Titan Xp	BW_S10
Numerical Type	Float32	BFP (1s.5e.2m)
Peak TFLOPS	12.1	48.0
TDP (W)	250	125
Process	TSMC 16nm	Intel 14nm

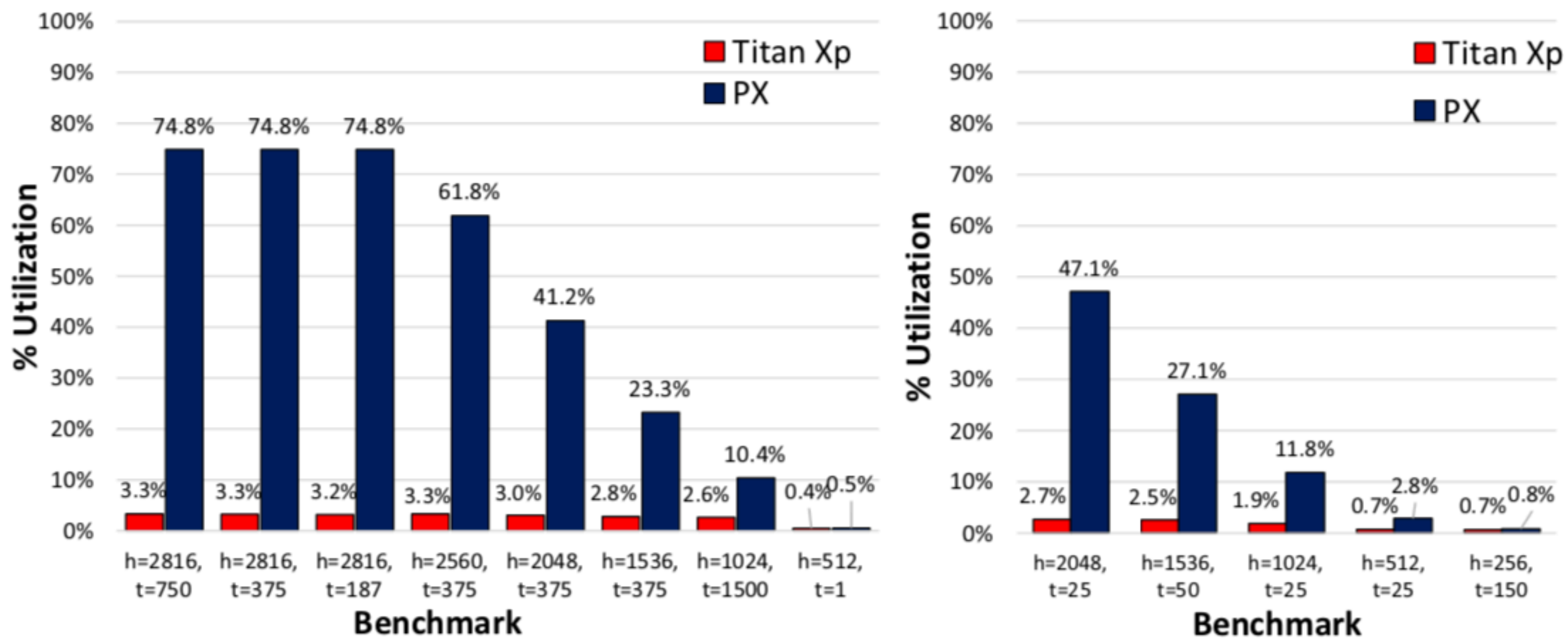
TABLE IV
EXPERIMENT HARDWARE SPECIFICATIONS

Benchmark	Device	Latency (ms)	TFLOPS	% Utilization
GRU h=2816 t=750	SDM	1.581	-	-
	BW	1.987	35.92	74.8
	Titan Xp	178.60	0.40	3.3
GRU h=2560 t=375	SDM	0.661	-	-
	BW	0.993	29.69	61.8
	Titan Xp	74.62	0.40	3.3
GRU h=2048 t=375	SDM	0.438	-	-
	BW	0.954	19.79	41.2
	Titan Xp	51.59	0.37	3.0
GRU h=1536 t=375	SDM	0.266	-	-
	BW	0.951	11.17	23.3
	Titan Xp	31.73	0.33	2.8
GRU h=1024 t=1500	SDM	0.558	-	-
	BW	3.792	4.98	10.4
	Titan Xp	59.51	0.32	2.6
GRU h=512 t=1	SDM	0.00017	-	-
	BW	0.013	0.25	0.5
	Titan Xp	0.06	0.05	0.4
LSTM h=2048 t=25	SDM	0.037	-	-
	BW	0.074	22.62	47.1
	Titan Xp	5.27	0.32	2.7
LSTM h=1536 t=50	SDM	0.043	-	-
	BW	0.145	13.01	27.1
	Titan Xp	6.20	0.30	2.5
LSTM h=1024 t=25	SDM	0.011	-	-
	BW	0.074	5.68	11.8
	Titan Xp	1.87	0.22	1.9
LSTM h=512 t=25	SDM	0.0038	-	-
	BW	0.077	1.37	2.8
	Titan Xp	1.26	0.08	0.7
LSTM h=256 t=150	SDM	0.0126	-	-
	BW	0.425	0.37	0.8
	Titan Xp	1.99	0.08	0.7

TABLE V
DEEPBENCH RNN INFERENCE PERFORMANCE RESULTS

Note that this is comparing 32-bit IEEE floating point to an 8-bit custom floating point format

Utilization Comparison



(a) GRU (b) LSTM

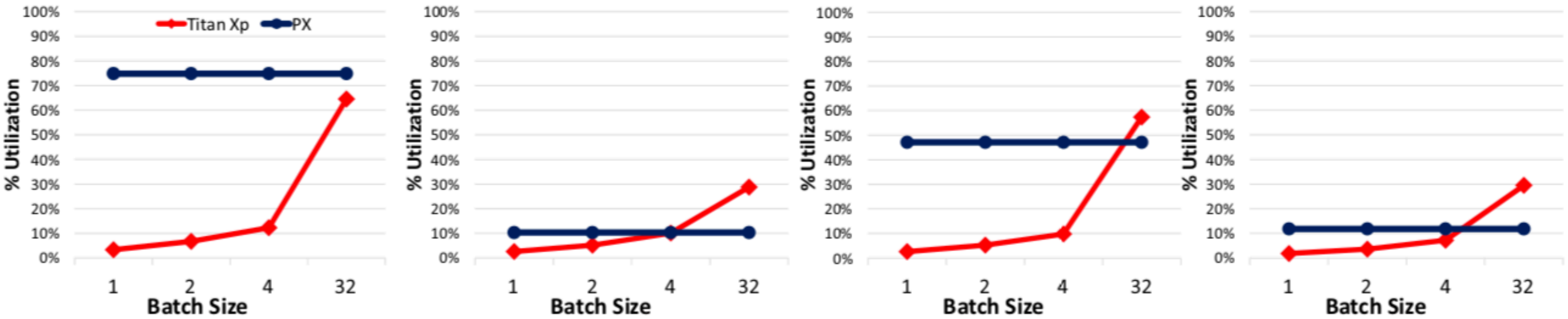
Fig. 7. Hardware utilization across DeepBench RNN inference experiments. h = hidden dimension, t = time steps

ResNet50 2D Classifier Benchmark

TABLE VI
THE BRAINWAVE NPU ON ARRIA 10 ACHIEVES COMPETITIVE THROUGHPUT AND LATENCY TO AN NVIDIA P40 GPU AT BATCH SIZE 1 ON A RESNET-50-BASED IMAGE FEATURIZER.

	Nvidia P40	BW_CNN_A10
Technology node	16nm TSMC	20nm TSMC
Framework	TF 1.5 + TensorRT 4	TF + BW
Precision	INT8	BFP (1s.5e.5m)
IPS (batch 1)	461	559
Latency (batch 1)	2.17 ms	1.8 ms

Batch Size Scaling



(a) GRU-2816 (b) GRU-1024 (c) LSTM-2048 (d) LSTM-1024

Fig. 8. Utilization scaling with increasing batch sizes.

Discussion