



Efficiently Training CNN with FPGA

Presented By

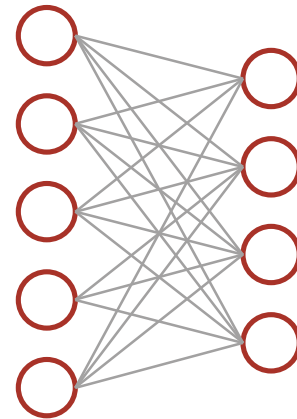
Yu Wang

Associate Professor, Dept. of E.E., Tsinghua University

Co-founder of DeePhi Tech.



An Era of Deep Learning



Neural Network



Cloud service



ADAS



Smart phone

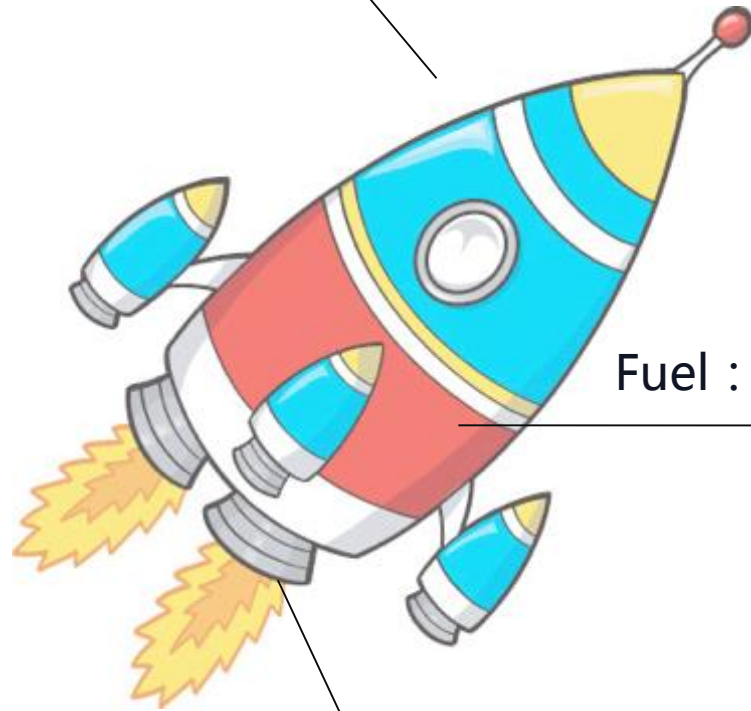


Camera

Everything & Everywhere

Two Stages in Deep Learning: Training and Inference

Rocket: Deep learning



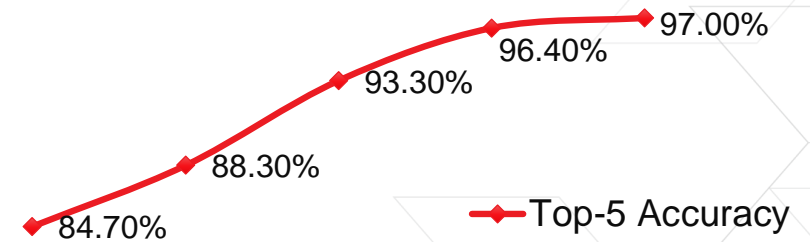
Fuel : Big data

Engine : Computing platform

According to Prof. Andrew Ng

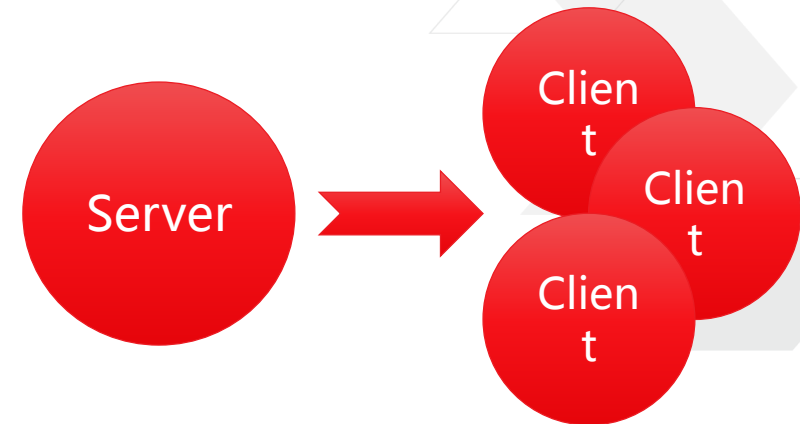
Training

– How accurate the model can be

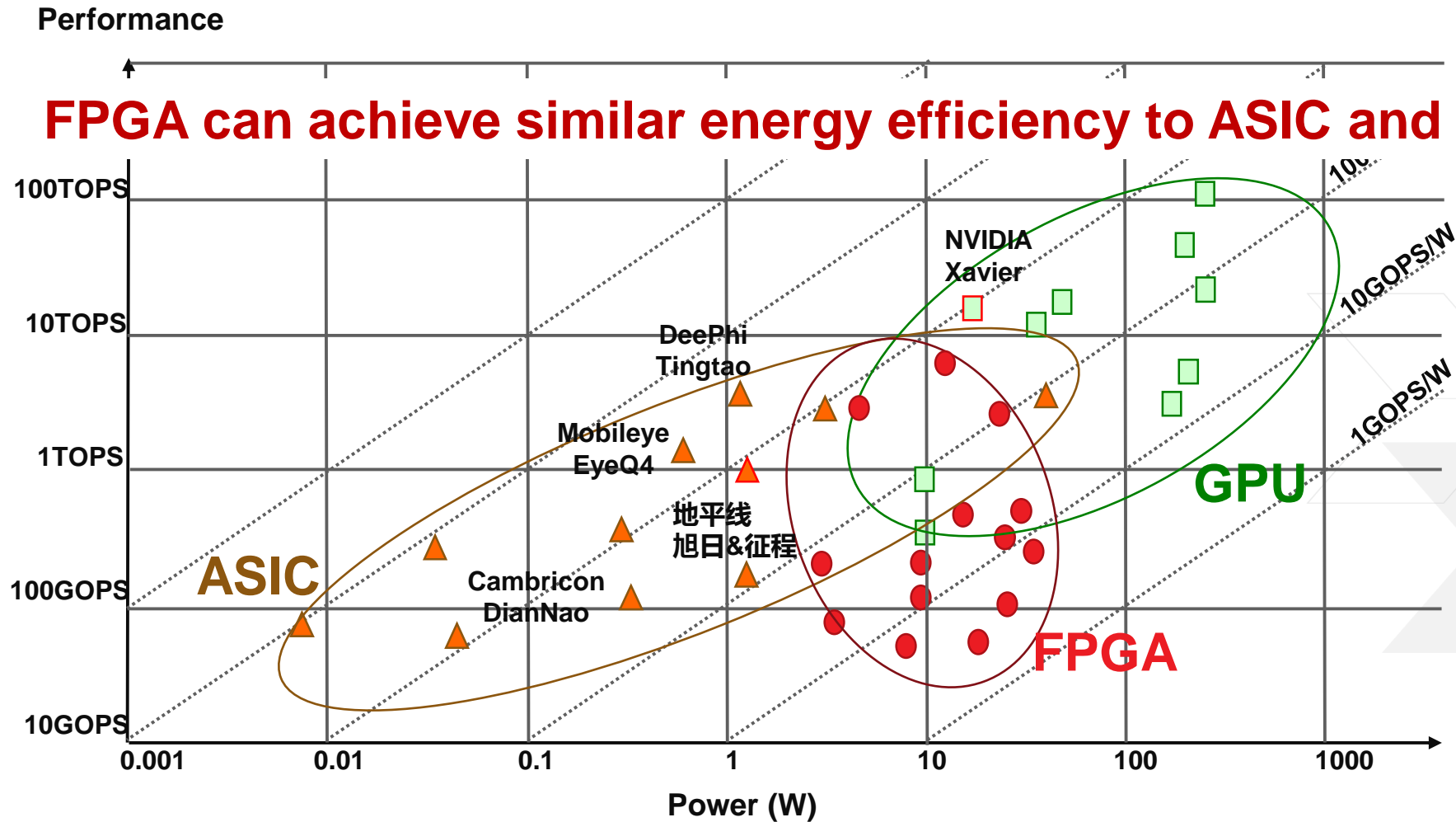


Inference

– How many applications can use DL

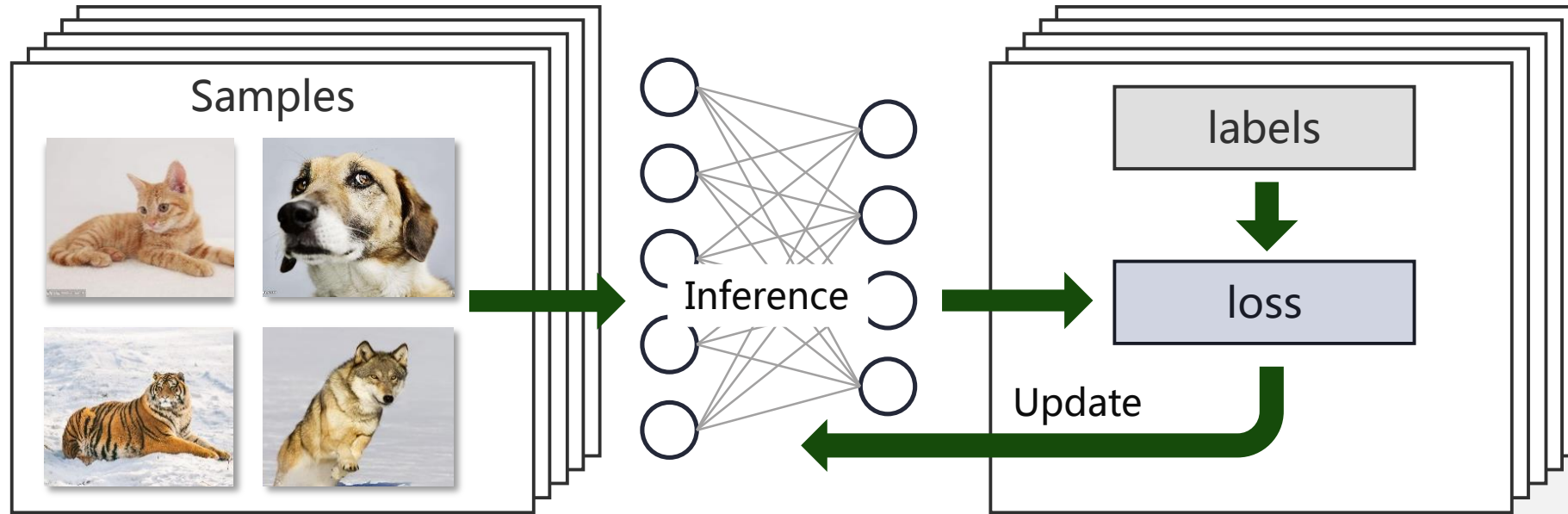


Efficient inference has been highly focused on



What about training?

> Training is also heavy work!



Training a VGG model on ImageNet dataset:

$$\text{Workload} = \left[\begin{array}{c} \# \text{Samples} \\ \sim 10^6 \end{array} \right] \times \left[\begin{array}{c} (\text{Inference} + \text{Update}) \\ \sim 10^{10} \text{ FLOPs} \end{array} \right] \times \left[\begin{array}{c} \# \text{Iterations} \\ \sim 100 \end{array} \right]$$



× 4 × 60hr

NVIDIA Titan Xp (TDP 250W)

We also need energy efficient training

> For cloud services:

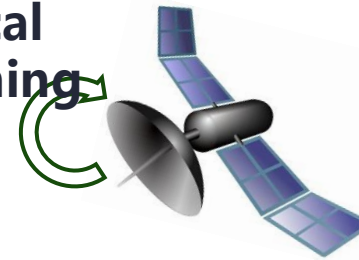
>> The building cost of a data center is about 10000-20000\$/kW¹



• For end applications:

- changing environment -> changing model
- Platform power limitation
- Car: 10-100w
- Satellite: hard to dissipate heat

local training



unstable



backend training



cloud training

privacy

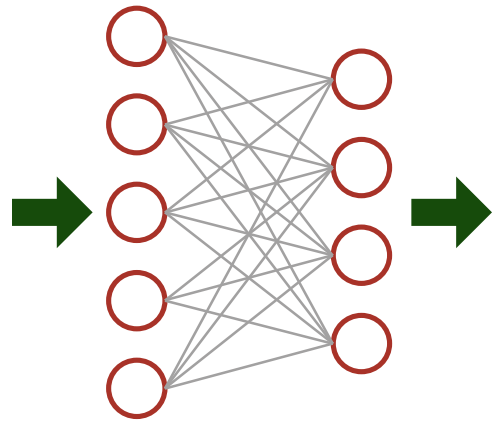


local training

¹ Kontorinis V, Zhang L E, Aksanli B, et al. Managing distributed ups energy for effective power capping in data centers[C]//Computer Architecture (ISCA), 2012 39th Annual International Symposium on. IEEE, 2012: 488-499.

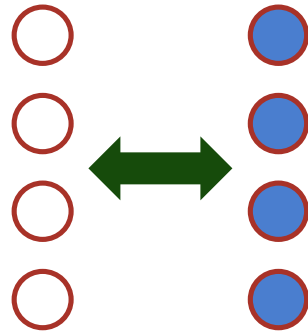
What does training do?

> Stochastic Gradient Descent (or other variants)



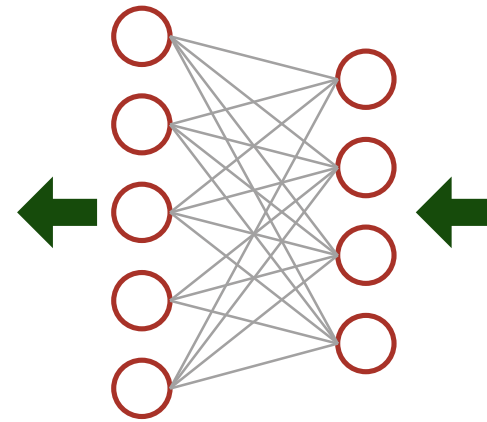
Inference

$$y = W \cdot x$$



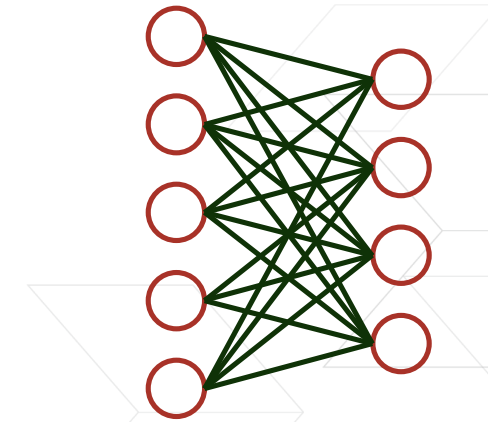
Calculate the error of output

$$\delta y$$



Back-propagate

$$\delta x = \delta y \cdot W'$$

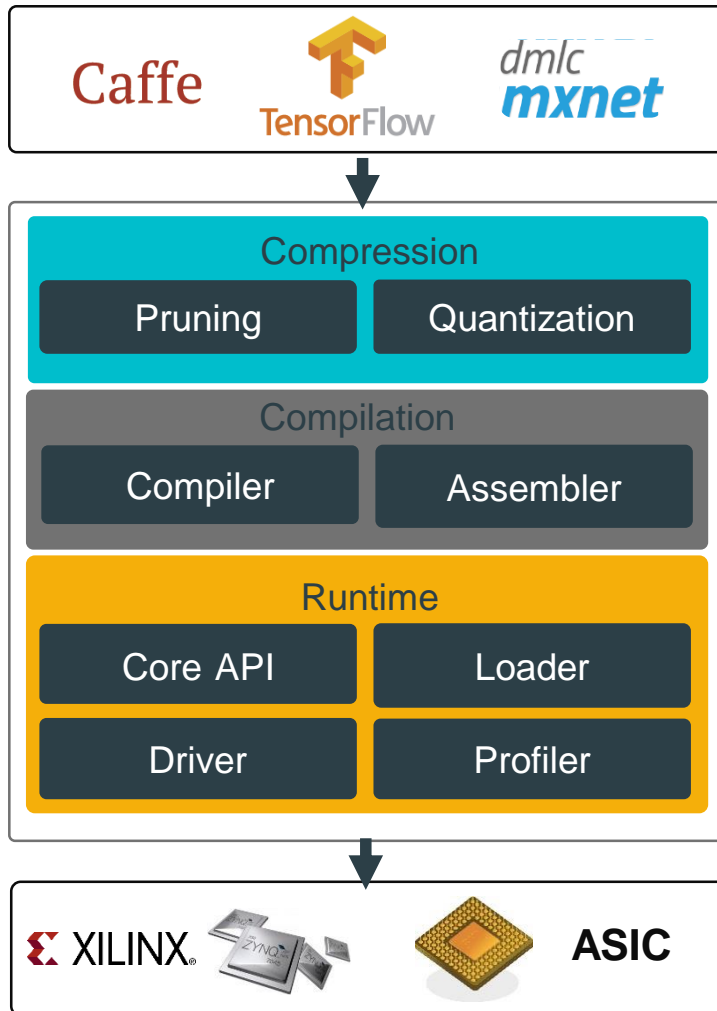


Update

$$\delta W = \delta y \cdot x'$$
$$W = W + \lambda \cdot \delta W$$

- Similar to inference, the main workload of training is still **sum of product**
- Maybe we can do the same things as for inference

What we have done with inference



DL Framework



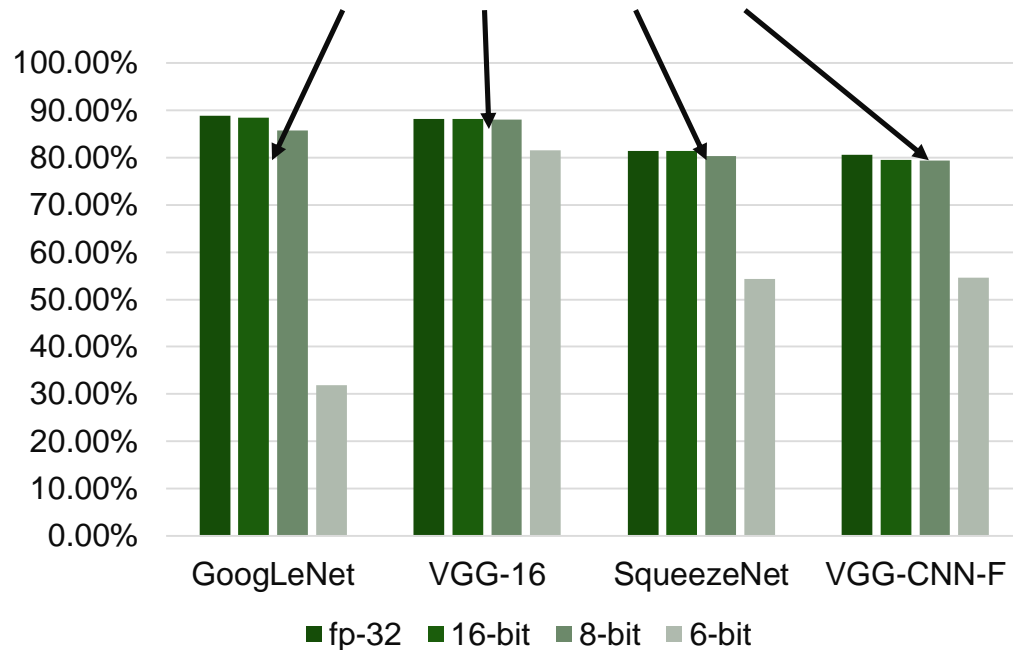
DPU Platform

Software-Hardware
Co-design

Techniques for inference: Software-Hardware Co-design

> Quantization

Negligible accuracy loss with 8-bit fixed-point weight & activation¹



Save FPGA resources

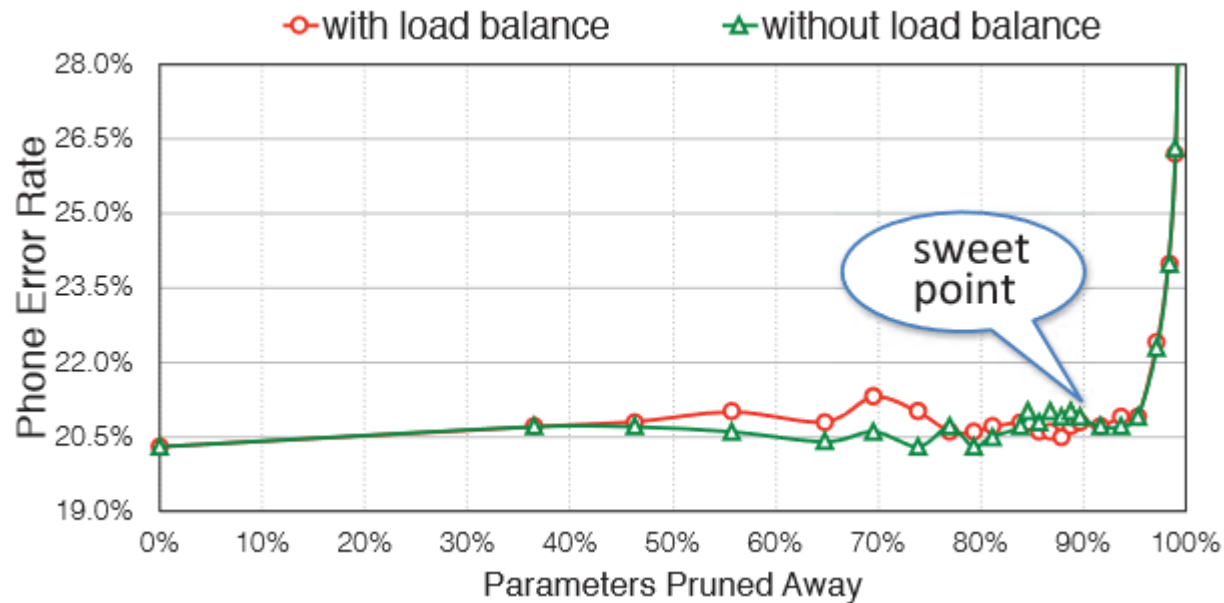
	Xilinx Logic				Xilinx DSP		
	multiplier		adder		multiply & add		
	LUT	FF	LUT	FF	LUT	FF	DSP
fp32	708	858	430	749	800	1284	2
fp16	221	303	211	337	451	686	1
fixed32	1112	1143	32	32	111	64	4
fixed16	289	301	16	16	0	0	1
fixed8	75	80	8	8	0	0	1
fixed4	17	20	4	4	0	0	1

¹ Guo K, Han S, Yao S, et al. Software-Hardware Codesign for Efficient Neural Network Acceleration[J]. IEEE Micro, 2017, 37(2): 18-25.

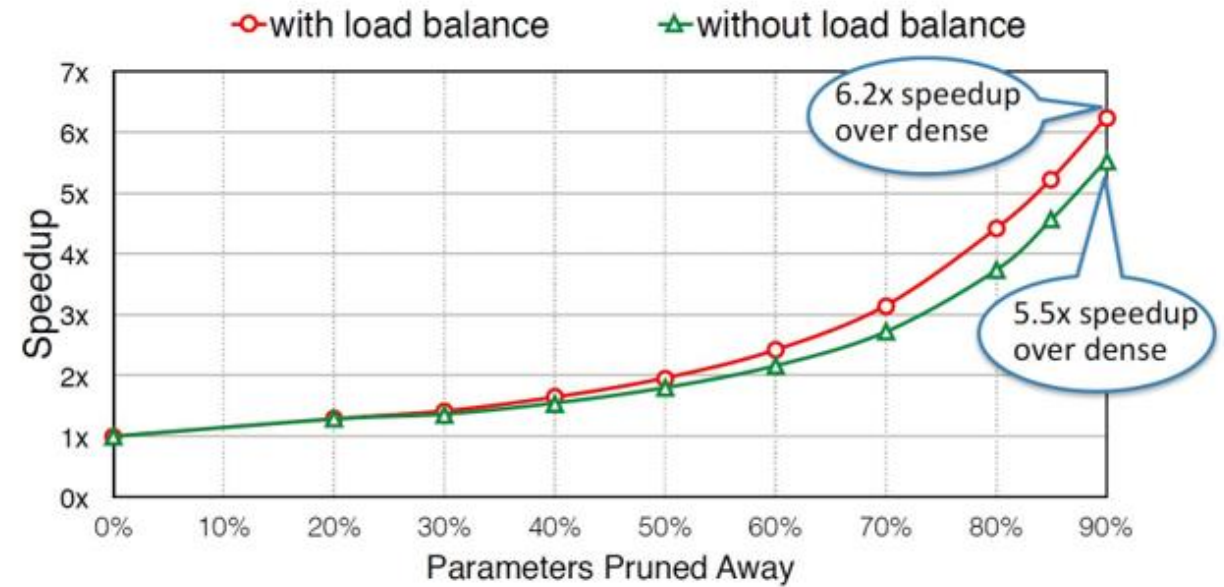
Techniques for inference: Software-Hardware Co-design

> Sparsification¹

Negligible WER increase with 10% weights



6x acceleration with customized hardware



¹ Han S, Kang J, Mao H, et al. ESE: Efficient speech recognition engine with sparse lstm on fpga[C]// ISFPGA, 2017: 75-84.

Q: Can we take everything from
inference to **training**?

A: Things are different!

Challenges

> Quantization for training is more difficult than for inference¹

Method	k_W	k_A	k_G	k_E	Opt	BN	MNIST	SVHN	CIFAR10	ImageNet
BC	1	32	32	32	Adam	✓	1.29	2.30	9.90	-
BNN	1	1	32	32	Adam	✓	0.96	2.53	10.15	-
BWN ¹	1	32	32	32	withM	✓	-	-	-	43.2/20.6
XNOR	1	1	32	32	Adam	✓	-	-	-	55.8/30.8
TWN	2	32	32	32	withM	✓	0.65	-	7.44	34.7/13.8
TTQ	2	32	32	32	Adam	✓	-	-	6.44	42.5/20.3
DoReFa ²	8	8	32	8	Adam	✓	-	2.30	-	47.0/ -
TernGrad ³	32	32	2	32	Adam	✓	-	-	14.36	42.4/19.5
WAGE	2	8	8	8	SGD	✗	0.40	1.92	6.78	51.6/27.8

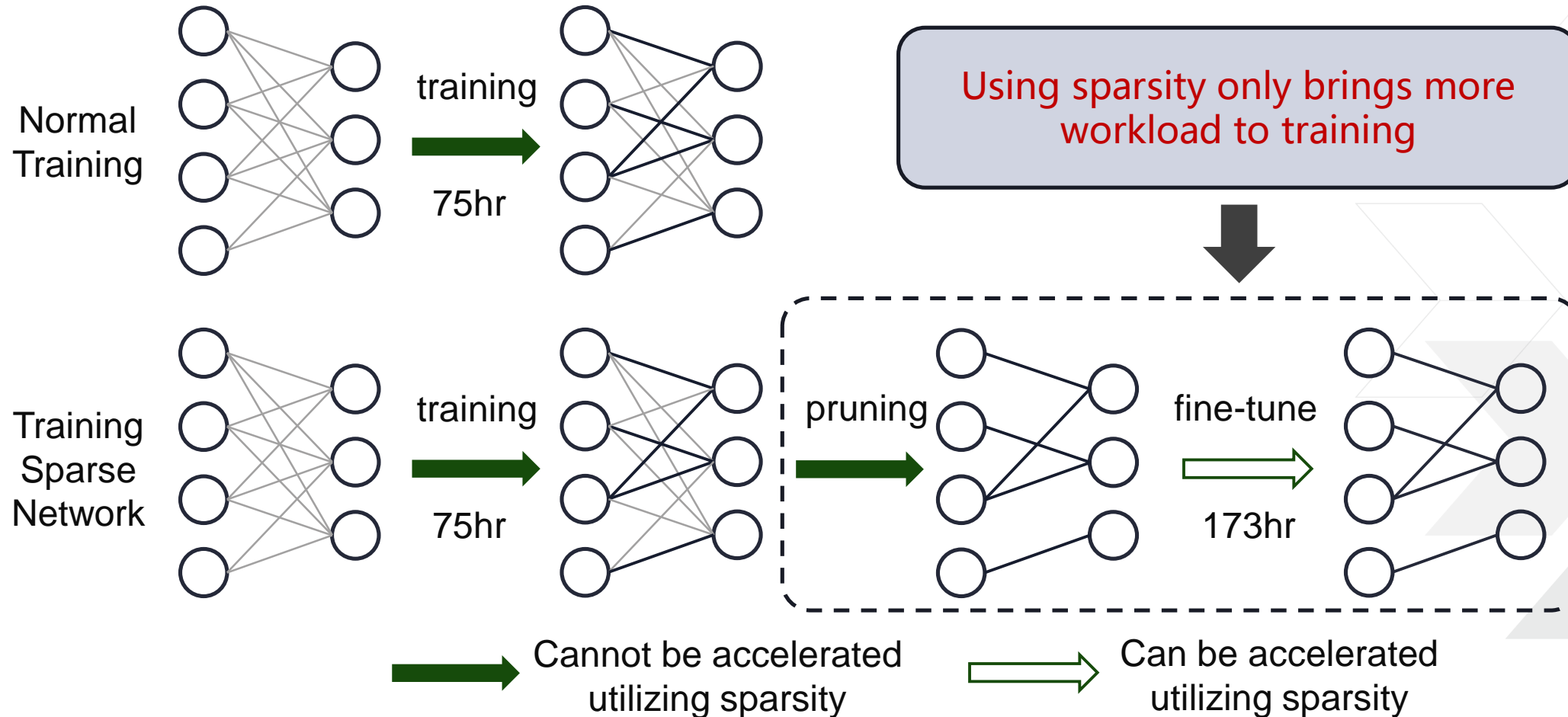
Baseline : 36.7/15.3

High accuracy v.s. Narrow bitwidth

¹ Wu S, Li G, Chen F, et al. Training and Inference with Integers in Deep Neural Networks[J]. arXiv preprint arXiv:1802.04680, 2018.

Challenges

> Pruning is applied when the network is well trained¹

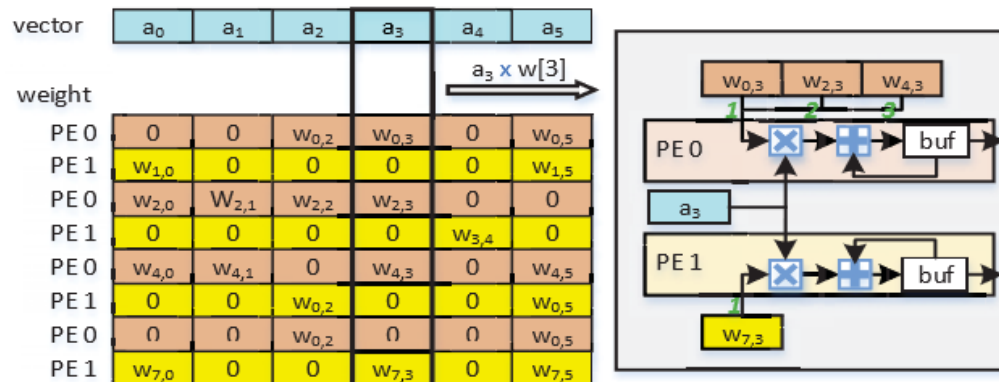


¹ Han S, Pool J, Tran J, et al. Learning both weights and connections for efficient neural network[C]/NIPS. 2015: 1135-1143.

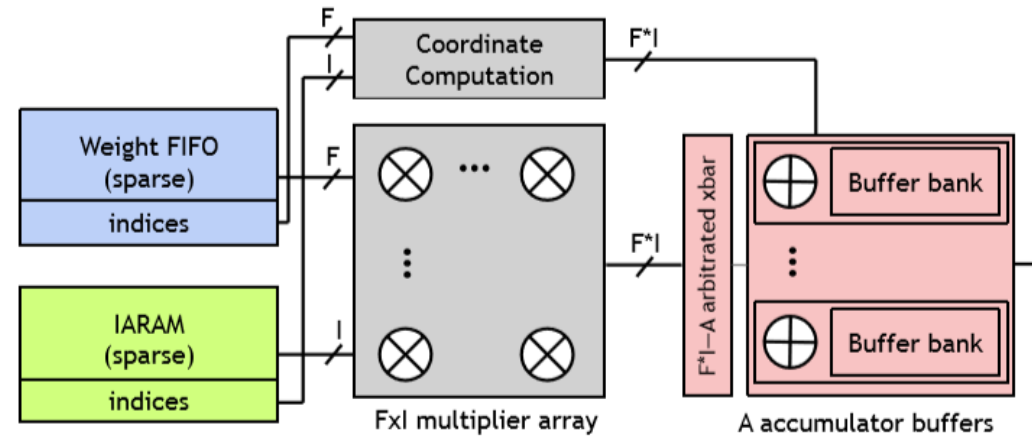
Opportunity

> Previous hardware only utilizes operand sparsity

- >> $C = \sum AB$ (A and B can be sparse)
- >> Apply to inference and back-propagate phases



ESE¹



SCNN²

• We should also utilize **result sparsity**

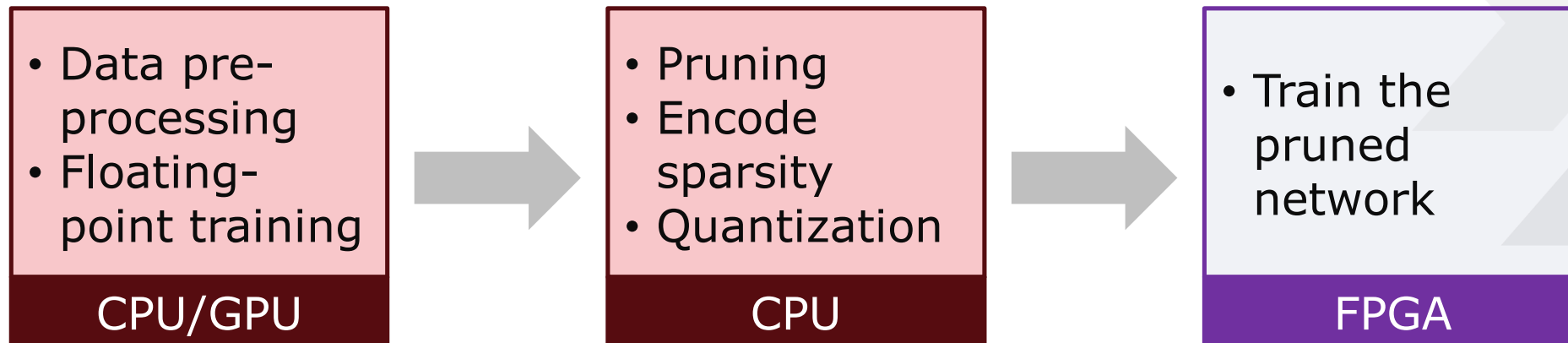
- $C = \sum AB$ (C is sparse)
- Apply to update phase: $\delta W = \delta y \cdot x'$

¹ Han S, Kang J, Mao H, et al. ESE: Efficient speech recognition engine with sparse lstm on fpga[C]// ISFPGA, 2017: 75-84.

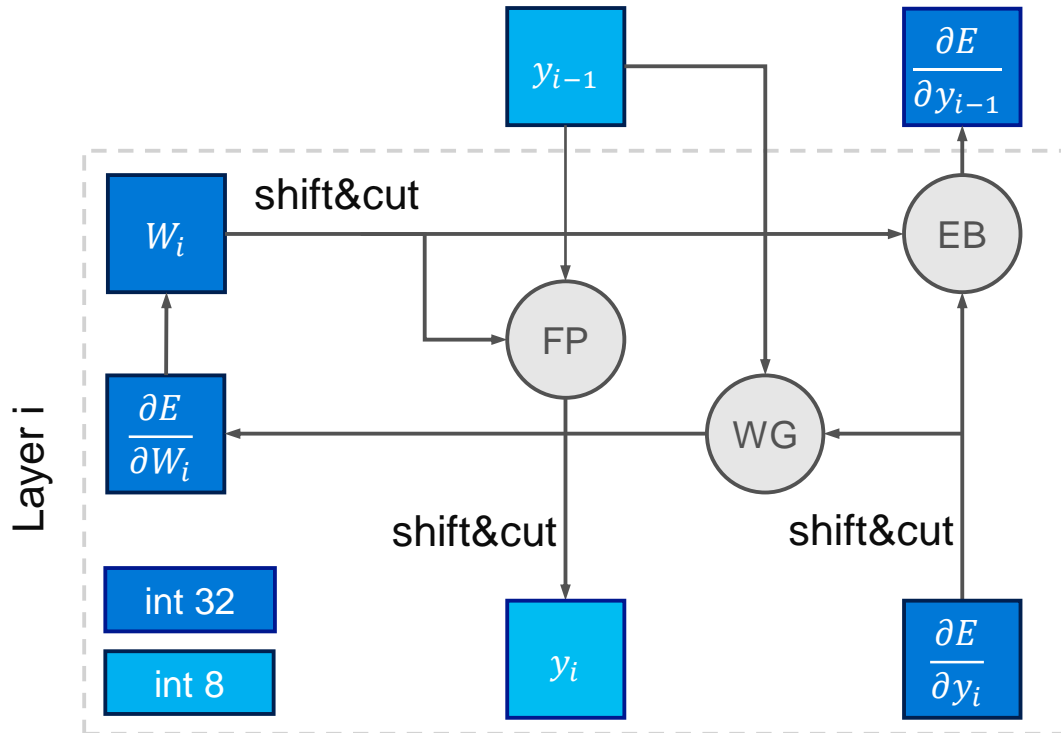
² Parashar A, Rhu M, Mukkara A, et al. Scnn: An accelerator for compressed-sparse convolutional neural networks[C]// ISCA, 2017: 27-40.

Our Solution

- 1. A training process with fixed-point gradient, activation, weight**
 - >> 32-bit float multiplication -> 8-bit fixed-point multiplication
 - >> 8x energy saving
- 2. Pruning before the training process converge**
 - >> Reduce floating point training process to 1/3
- 3. Customized FPGA accelerator design to accelerate the training of the sparse model.**
 - >> Potentially more than 3x speed up for the largest layers



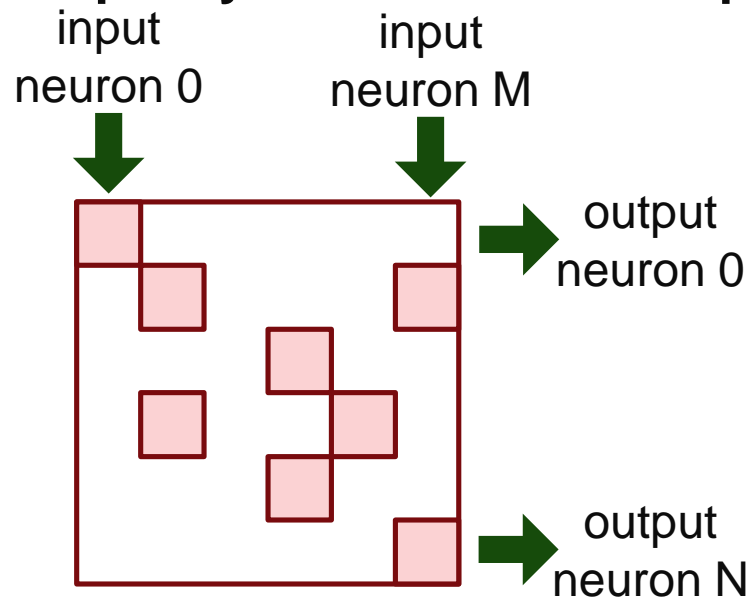
Training with fixed-point data



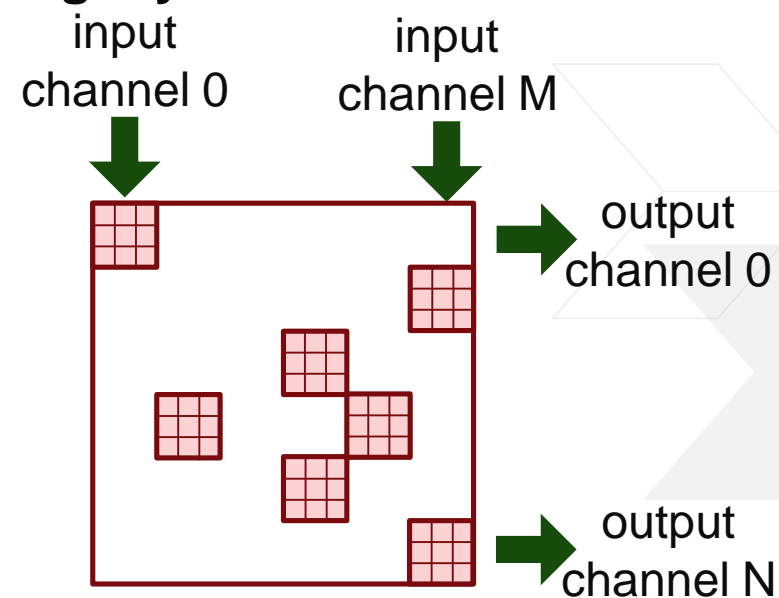
- > Replace all the floating point data with fixed-point data
- > Short bit-width for MAC: reduce computation cost
- > Long bit-width for weight storage: able to accumulate small gradients

Pruning before convergence

- > 2D-kernel-wise pruning for CONV layers
- > Normal pruning for FC layers
- > For the simplicity of hardware and sparse coding style



MxN fully connected layer

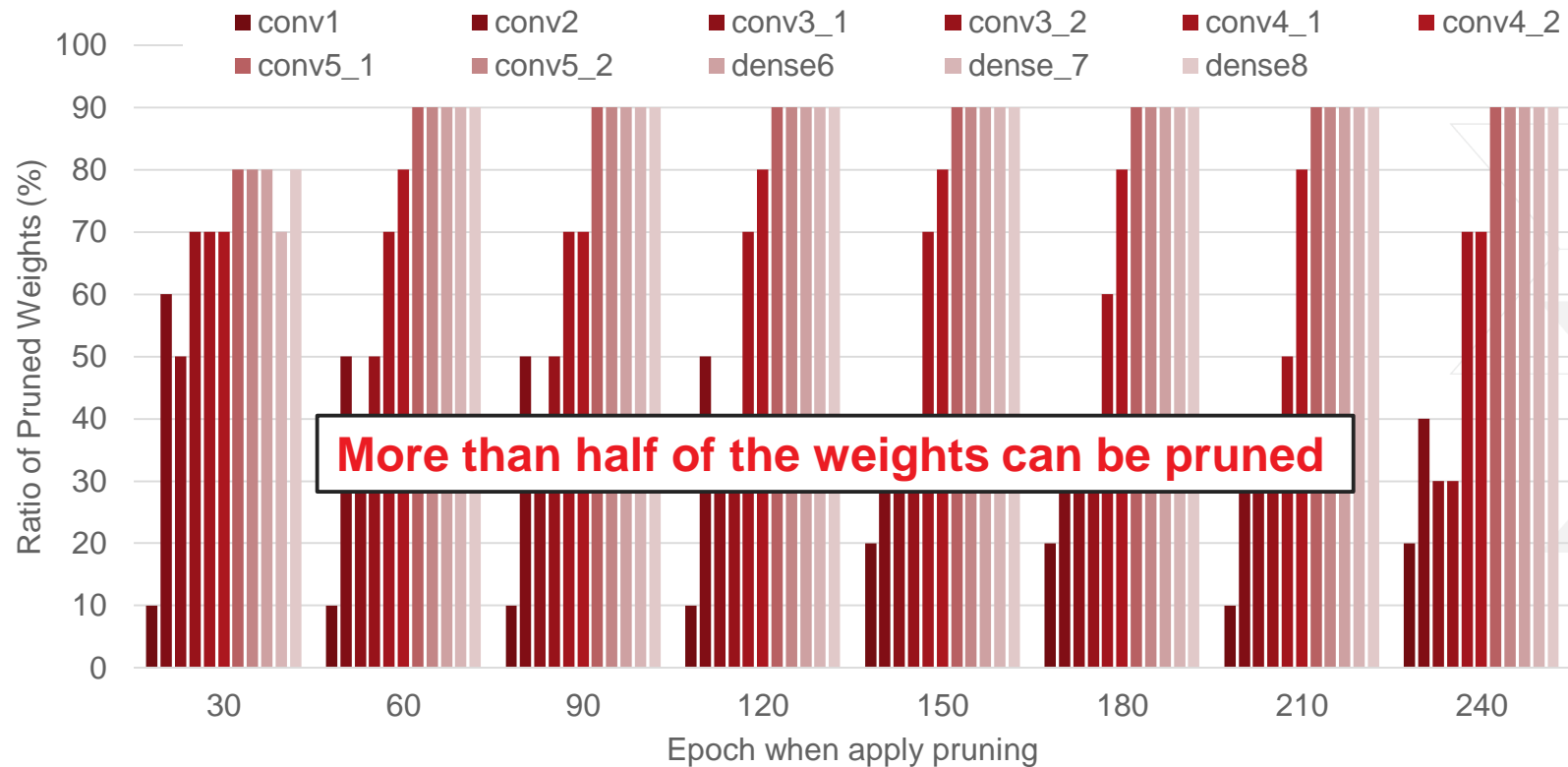


MxN channel convolution layer with 3x3 kernels

Pruning before convergence

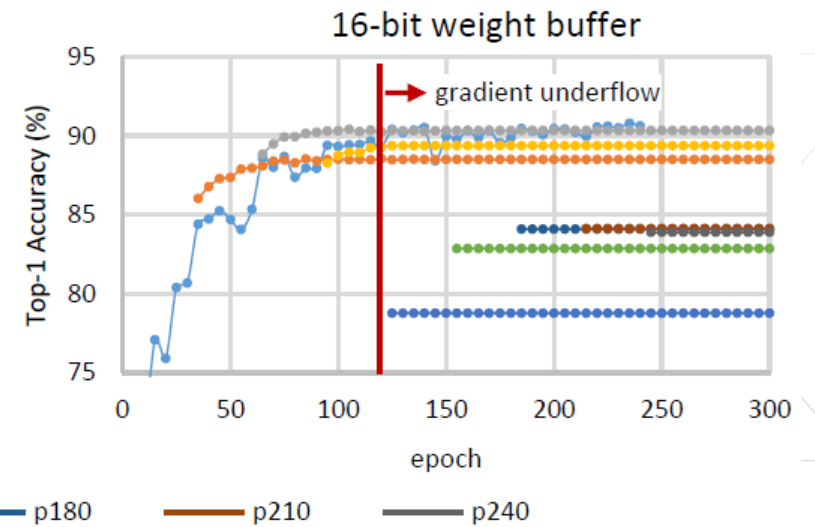
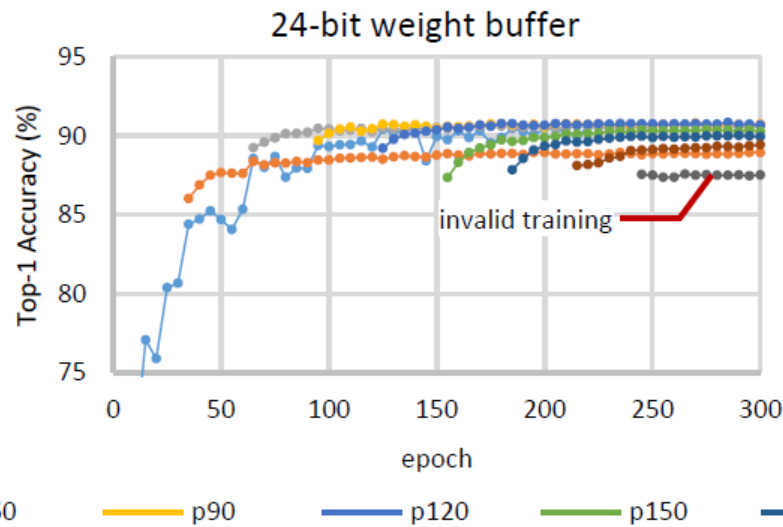
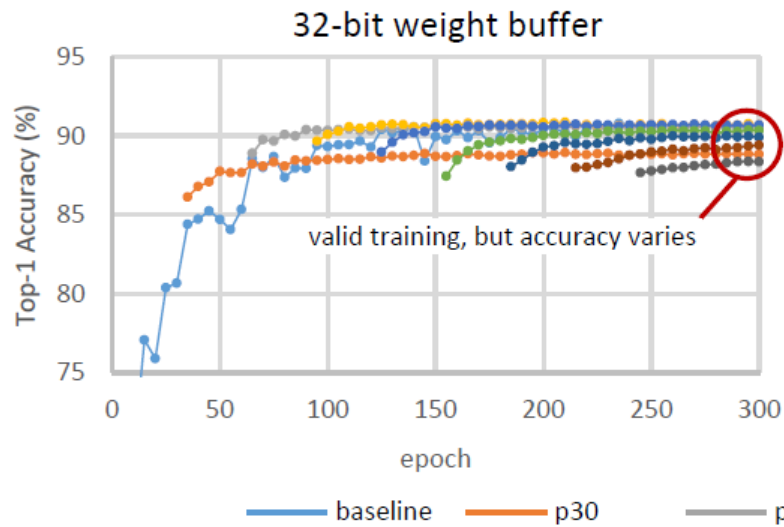
> Dataset: CIFAR-10; Network: VGG-11

> Prune standard: <1% accuracy loss

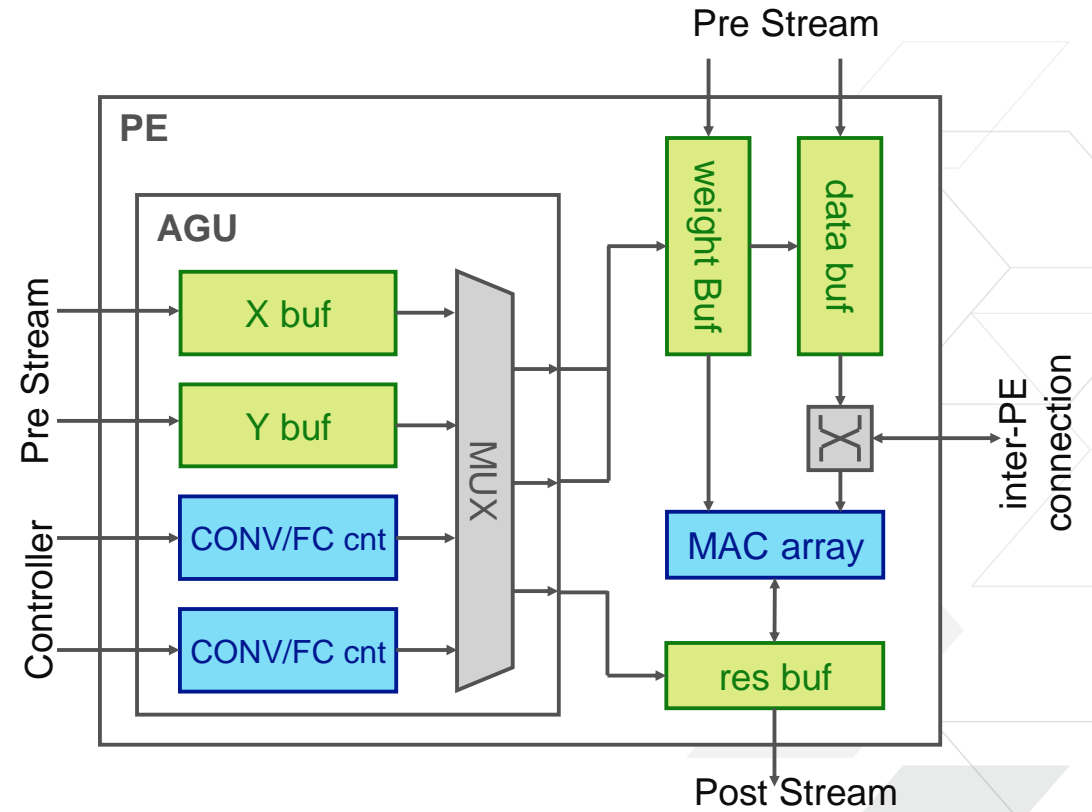
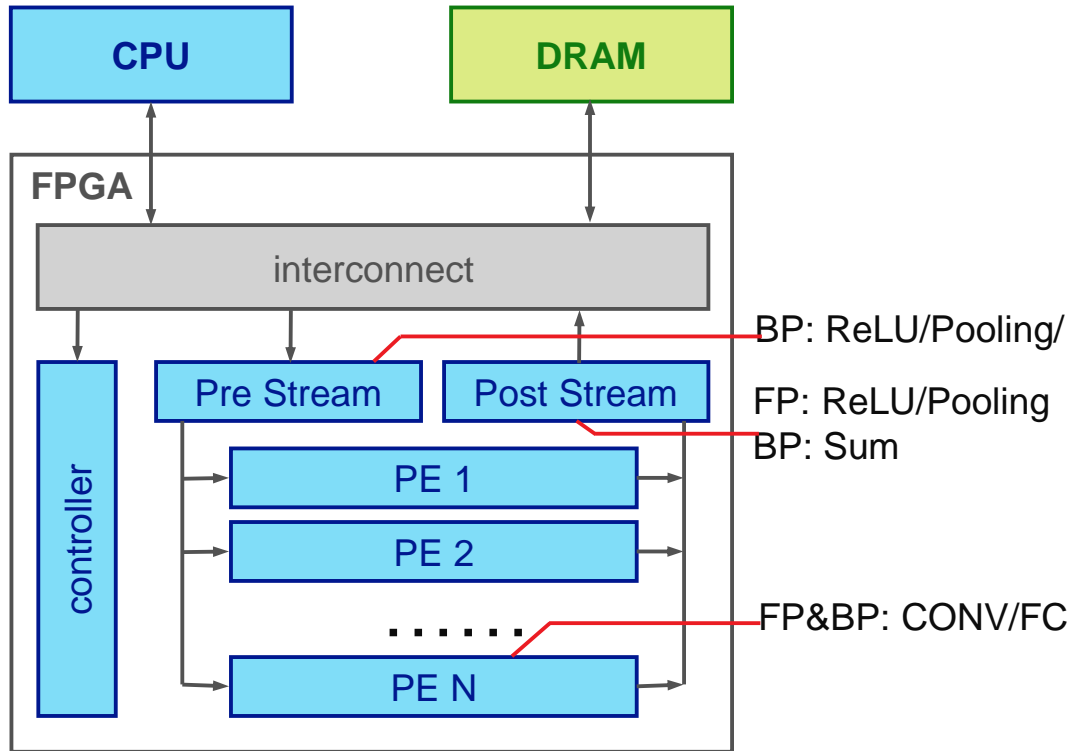


Pruning and Quantization

- > Dataset: CIFAR-10; Network: VGG-11
- > Weight: 32/24/16 bit; Activation: 8 bit
- > 24bit weight and pruning as early as 60 epochs is enough for a good training result



Hardware design



Parallelism

> Batch is welcomed for training

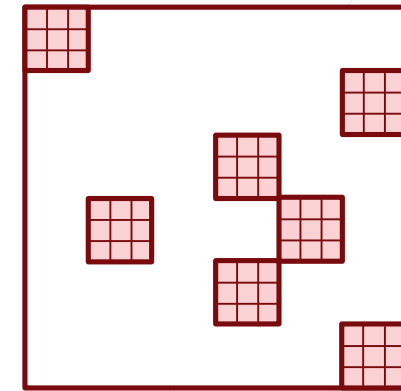
- ✓ Parallel batch process

> Input / Output channel

- ! Affected by sparsity
- ✓ Duplicate input channel and parallelize output channel

> Feature map pixel / Convolution kernel

- ! Loop size varies greatly
- ✓ Configurable parallelism

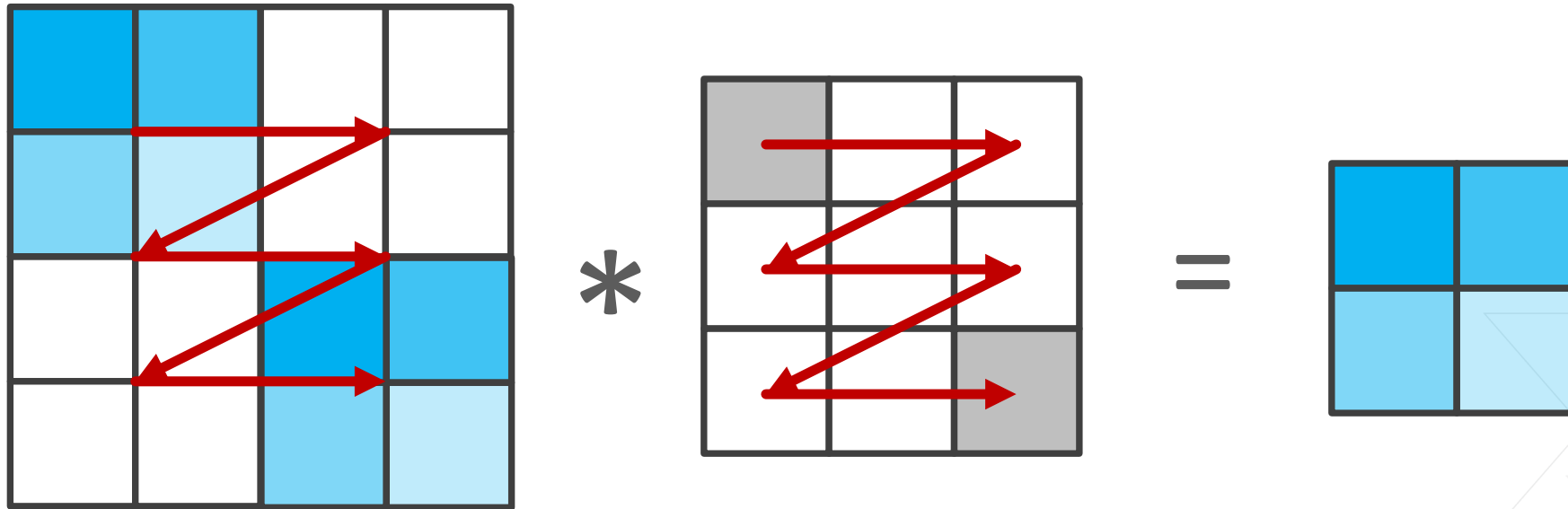


Inference: $F_j^l = \sum_{i=0}^{M-1} \text{conv2d}(F_i^{l-1}, K_{ij}) + b_j \quad j = 0, 1, \dots, N-1$

Update: $dK_{ij} = \text{conv2d}(F_i^{l-1}, dF_j^l) \quad \begin{matrix} j = 0, \dots, N-1 \\ i = 0, \dots, M-1 \end{matrix}$

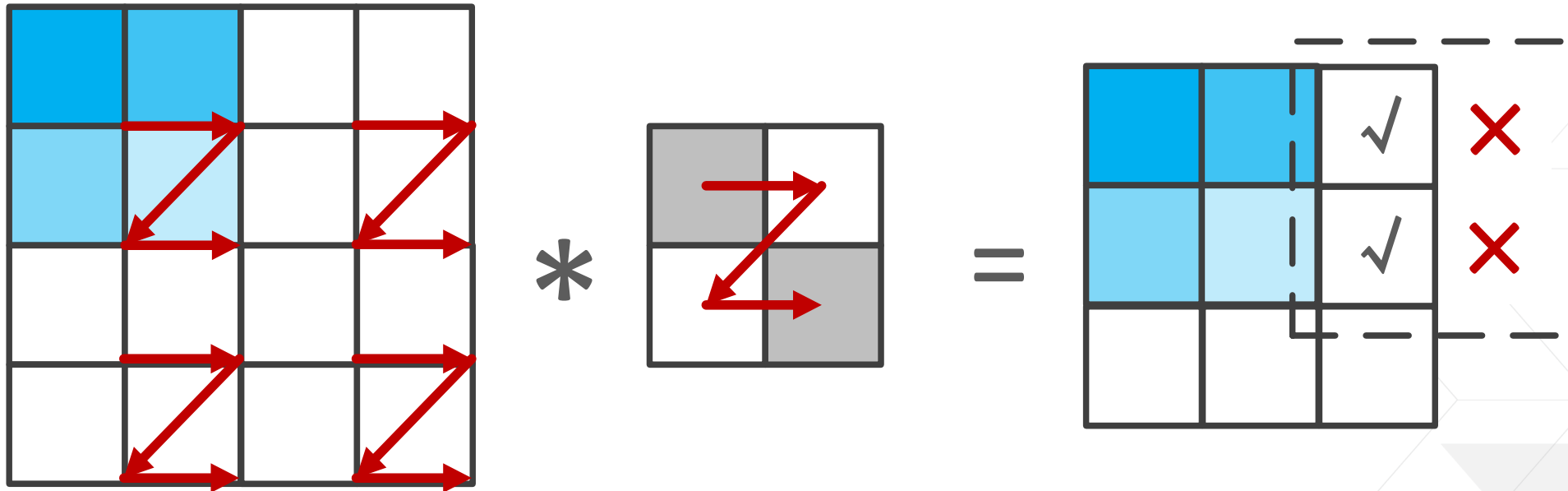
Diagram illustrating the flow of data and gradients. The word "large" is positioned above the inference equation, and "small" is positioned below the update equation. Red arrows point from "large" to the F_i^{l-1} term in the inference equation and from the dF_j^l term in the update equation to "small".

Configurable parallelism



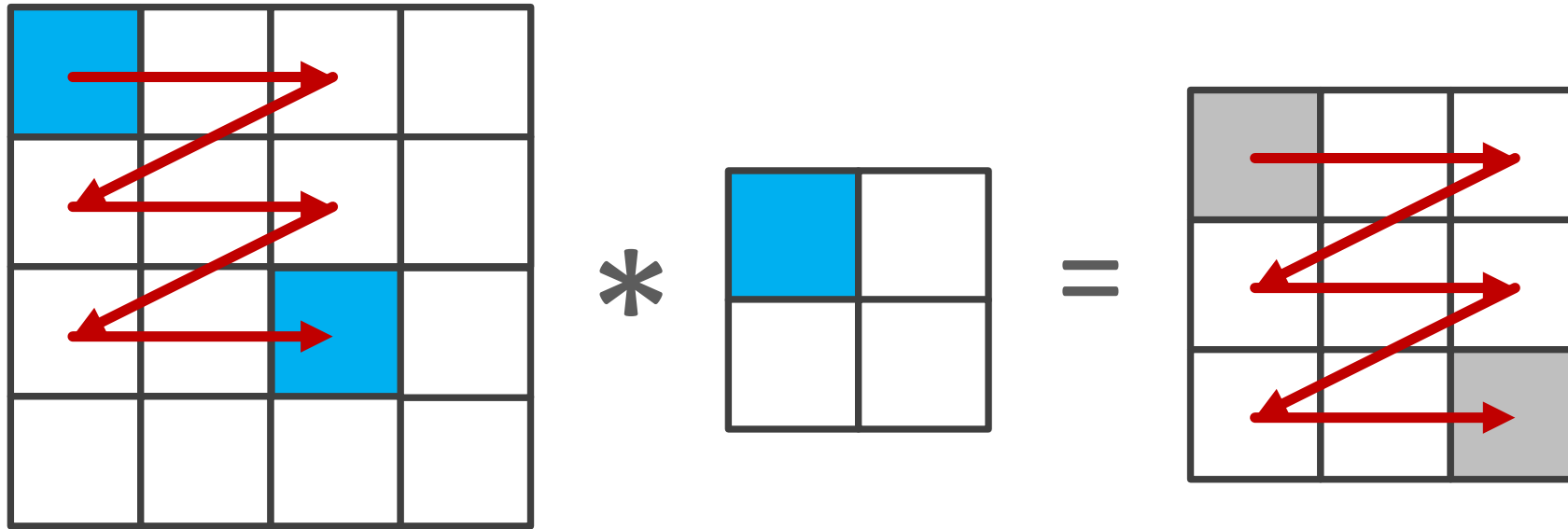
- > Each PE locally do multiply and accumulate
- > Inference: 4 PEs process 4 output pixels for 2D convolution in parallel
 - >> Data is shared within a group of PE to utilize the data locality

Configurable parallelism



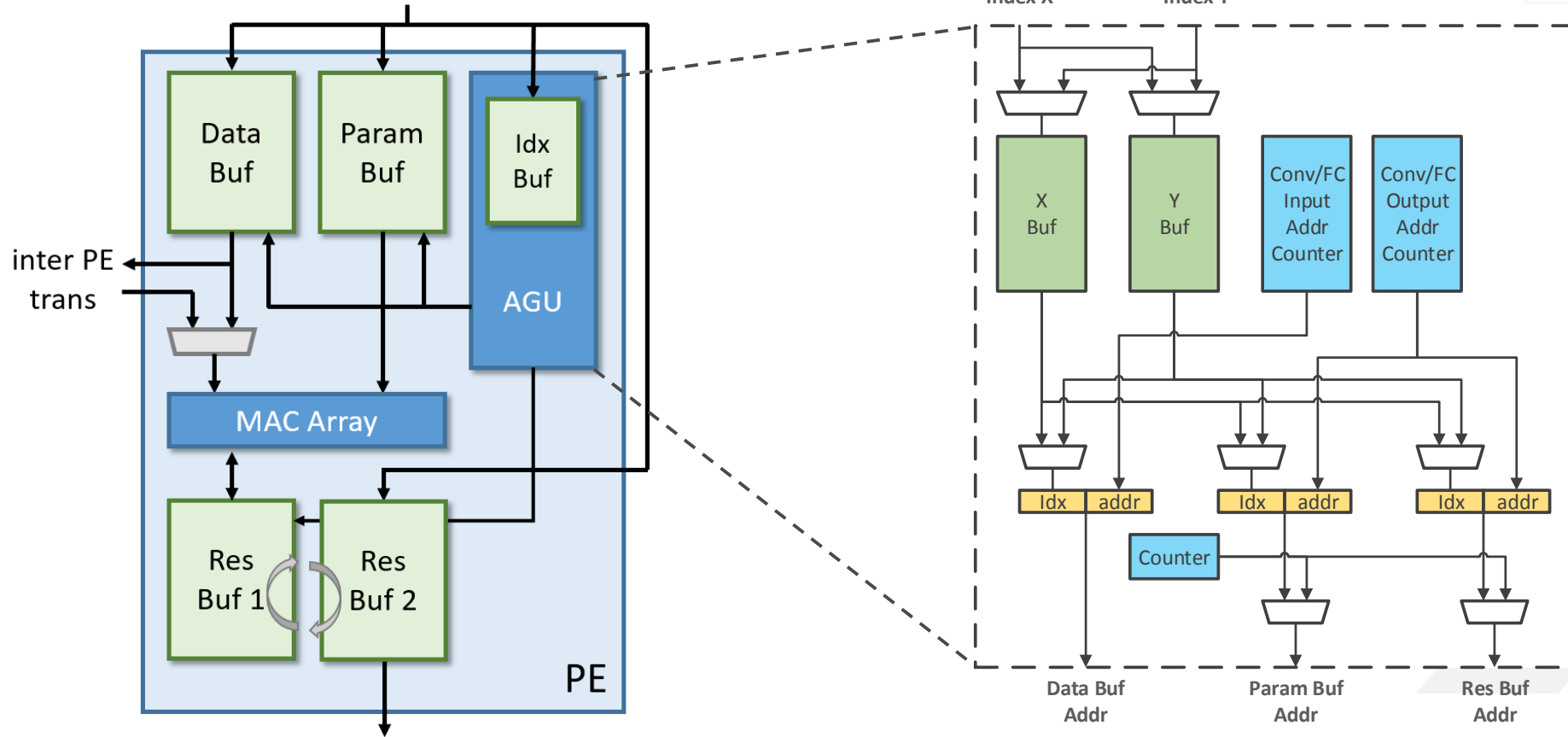
- Each PE locally do multiply and accumulate
- Inference: 4 PEs process 4 output pixels for 2D convolution in parallel
 - Data is shared within a group of PE to utilize the data locality
- Update: same with inference
 - Waste the computation units!

Configurable parallelism



- > **Each PE locally do multiply and accumulate**
- > **Inference: 4 PEs process 4 output pixels for 2D convolution in parallel**
 - >> Data is shared within a group of PE to utilize the data locality
- > **Update: 4PEs process a part of the gradient in parallel**
 - >> Partial gradients are summed in a successive module

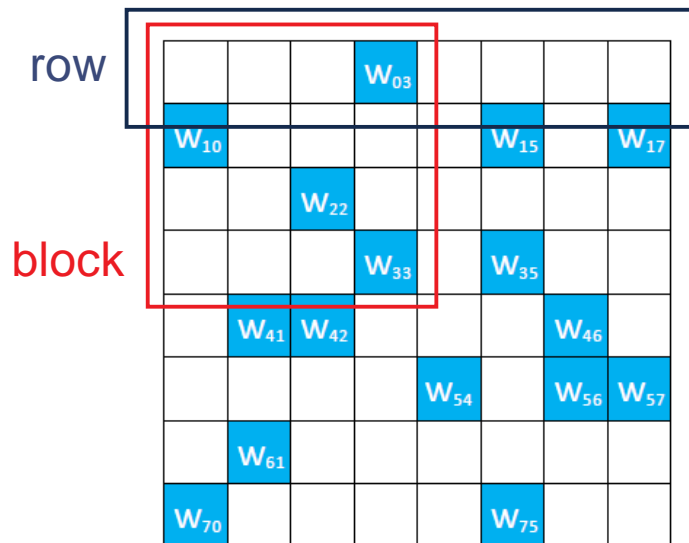
PE structure to support result sparsity



- > Random access buffer for both input and output data
- > Switch and MUX for index and address generation unit

Use Compressed Sparse Block (CSB) format

- > **FC layers:** split the weight matrix into blocks and encode each element in a block with a 2D coordinate.
- > **CONV layers:** we split the kernels into channel blocks and encode each 2d kernel in a block with a 2D coordinate.
- > Transpose by switch both the 2D coordinate in hardware and the block order in software



CSR format

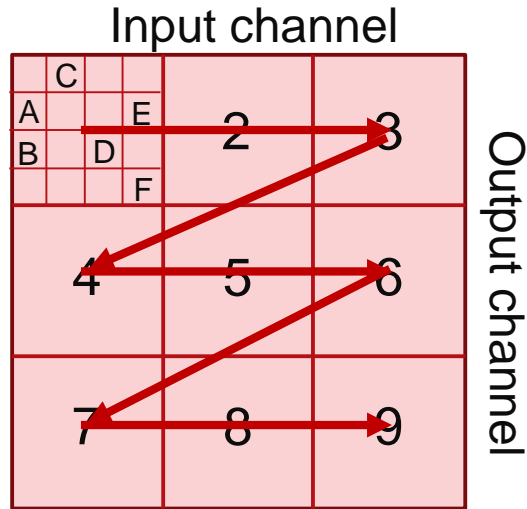
data	W ₁₀	W ₇₀	W ₄₁	W ₆₁	W ₂₂	W ₄₂	W ₀₃	W ₃₃	W ₅₄	W ₁₅	W ₃₅	W ₇₅	W ₄₆	W ₅₆	W ₁₇	W ₅₇
row idx	1	7	4	6	2	4	0	3	5	1	3	7	4	5	1	5
col ptr	0	2	4	6	8	9	12	14								

CSB format

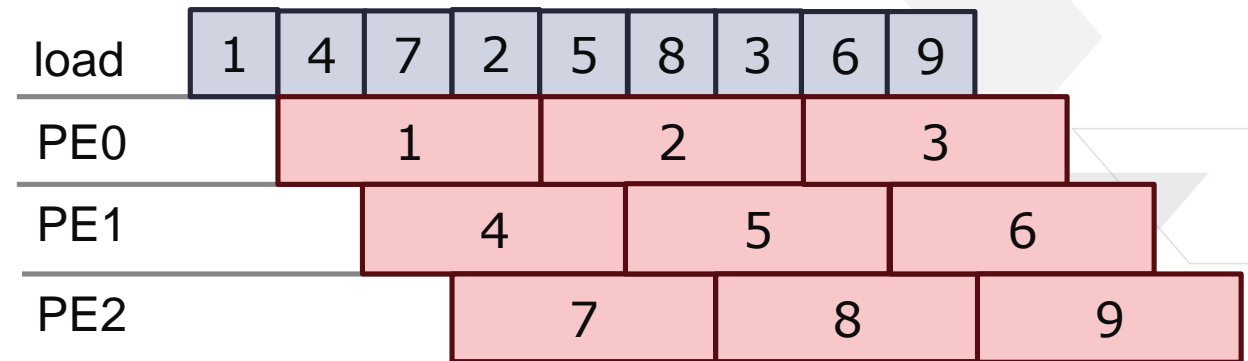
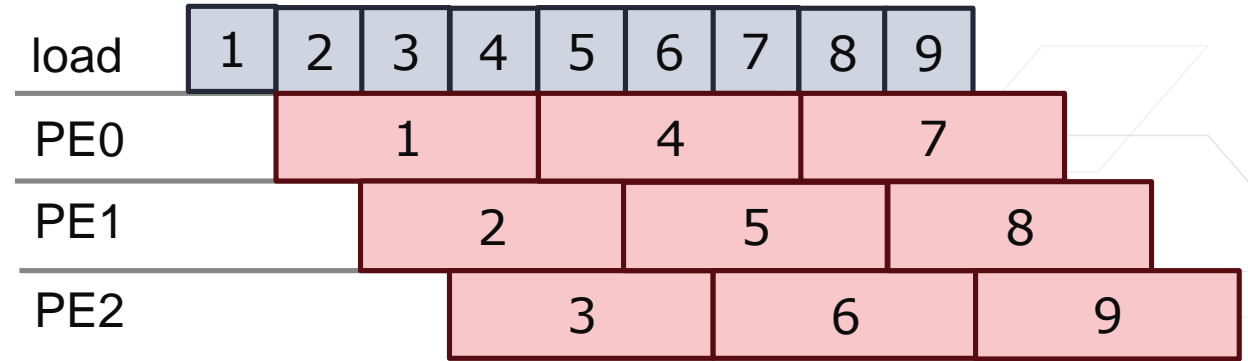
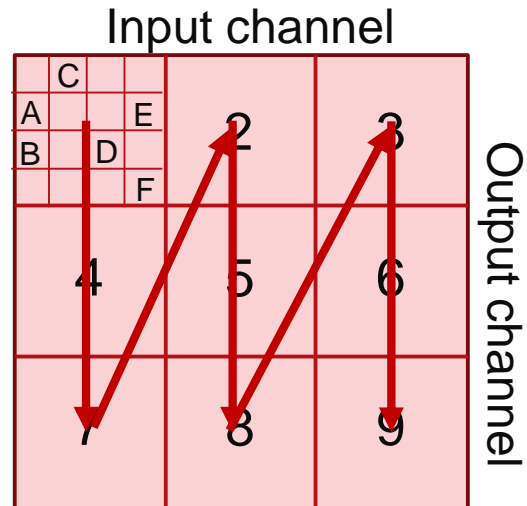
data	W ₁₀	W ₂₂	W ₀₃	W ₃₃	W ₇₀	W ₄₁	W ₆₁	W ₄₂	W ₁₅	W ₃₅	W ₁₇	W ₅₄	W ₇₅	W ₄₆	W ₅₆	W ₅₇
row idx	1	2	0	3	3	0	2	0	1	3	1	1	3	0	1	1
col idx	0	2	3	3	0	1	1	2	1	1	3	0	1	2	2	3
blk ptr	0	4	8	11												

Scheduling

Inference:
Each PE calculates 4 output channels



Back-propagate:
Each PE calculates 4 input channels



Evaluation

> **FPGA Platform: KCU1500 development board**

> **Hardware Parameter:**

- >> 250MHz clock
- >> 32 PE, each with 32 MAC unit to process a batch of 32 images
- >> 2 DDR4-2400 external memory

Resource	LUT	Reg	Block RAM	DSP
Available	663360	1326720	2160	5520
Utilization	199111	249122	1060	1030
Ratio	30%	19%	49%	19%

> **Bounded by on-chip memory**

> **Accumulation buffer occupies most of the RAMs because:**

- >> Large bitwidth (32 compared with 8 for data and param)
- >> Ping-pong strategy to cover data transfer time
- >> Ultra-RAM may relief this problem

Evaluation

Computation Bounded

Memory Bounded

> Performance breakdown (simulation)

layer	Feed Forward (FF)					Neuron Gradient(NG)				Neuron Gradient(NG)			
	Comp. (GOP)	Time (us)	Perf. (GOP/s)	bound type	Utilize rate	Time (us)	Perf. (GOP/s)	bound type	Utilize rate	Time (us)	Perf. (GOP/s)	bound type	Utilize rate
conv1	0.11	733	154.4	B	27%	-	-	-	-	4158	27.2	B	5%
conv2	1.21	1487	812.2	C	95%	1487	812.5	C	95%	2804	430.8	B	50%
conv3_1	1.21	1696	712.4	C	97%	1694	713.0	C	97%	2477	487.7	B	67%
conv3_2	2.42	2877	839.7	C	98%	2876	840.1	C	98%	4398	549.3	B	64%
conv4_1	1.21	1217	992.3	C	97%	1217	992.9	C	97%	2686	449.8	B	44%
conv4_2	2.42	1457	1658.4	C	97%	1456	1659.2	C	97%	3933	614.2	B	36%
conv5_1	0.60	366	1651.7	B	32%	358	1687.7	B	33%	915	659.8	B	13%
conv5_2	0.60	365	1652.7	B	32%	358	1688.7	B	33%	915	660.0	B	13%
dense6	0.02	329	51.0	B	1%	328	51.1	B	1%	717	23.4	B	0.5%
dense7	0.02	329	51.0	B	1%	328	51.1	B	1%	717	23.4	B	0.5%
dense8	0.003	75	4.4	B	0.1%	74	4.4	B	0.1%	272	1.2	B	0.02%
total	9.81	10931	897.5	-	-	10988	892.8	-	-	23993	408.9	-	-

Evaluation

> Performance comparison with state-of-the-art CNN inference/training accelerators and GPU

Platform	TCAD 18	FPGA 17	ESE	FCNN		FPT17	FPDeep	Proposed		GPU Titan X
	XC7Z020	GX1150	KU060	Maxeler MPC-X		ZU19EG	VC709	KCU1500		GM200
Function	Inference	Inference	Inference	Inference	Training	Training	Training	Inference	Training	Training
Quantization	fixed 8	fixed 8/16	fixed 12	float 32	float 32	float 32	fixed 16	fixed 8	fixed 8/24	float32
Sparsity	No	No	Yes	No	No	No	No	Yes	Yes	No
Performance (GOP/s)	84.3	645.25	2516	62.06	7.01	86.12	1022	897.5	641.1	1252
							(Per FPGA)			
Power (W)	3.5	21.2	41	N.A.	27.3	14.2	32	29		150
Energy Eff. (GOP/s/W)	24.1	30.43	61.4	N.A.	0.27	6.05	31.97	30.95	22.11	8.4

Future work

> More functions for training

- >> BN layers, quantization for other optimizers

> Scaling up the design?

- >> Reduce memory consumption by optimizing memory design
- >> Lower down bandwidth requirement by optimizing scheduling
- >> We also have HBM

> Improve energy efficiency?

- >> More advanced training methods

> Evaluation with real training tasks

- >> We never know if the method applies to a new application



Adaptable.
Intelligent.

