Crossbar-Aligned & Integer-Only Neural Network Compression for Efficient In-Memory Acceleration

Shuo Huai[1,2], Di Liu[2], Xiangzhong Luo[1], Hui Chen[1], Weichen Liu[1*], and Ravi Subramaniam[3]

[1]School of Computer Science and Engineering, Nanyang Technological University, Singapore
[2]HP-NTU Digital Manufacturing Corporate Lab, Nanyang Technological University, Singapore
[3]Innovations and Experiences – Business Personal Systems, HP Inc., Palo Alto, California, USA

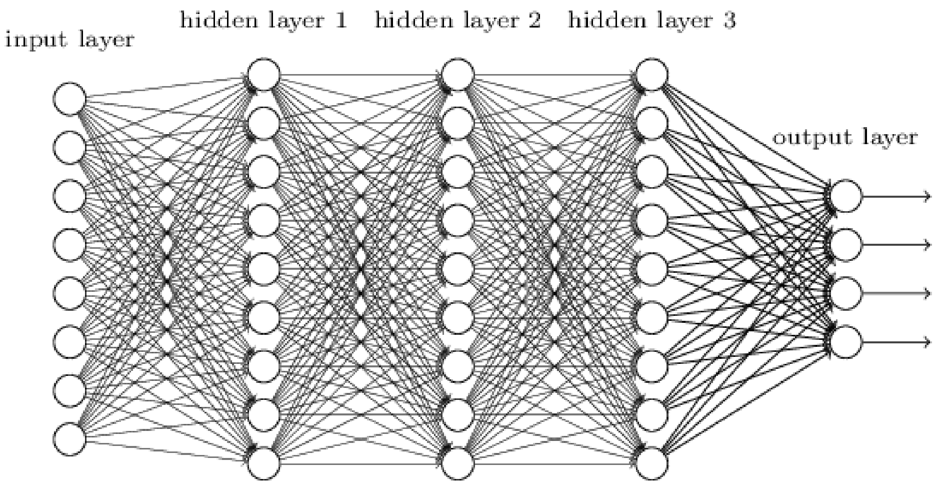
Outline

- Background & Motivation
- Crossbar-Aligned IMC Pruning
- Integer-Only IMC Quantization
- Co-optimization Framework
- Experimental evaluation
- Conclusion

Outline

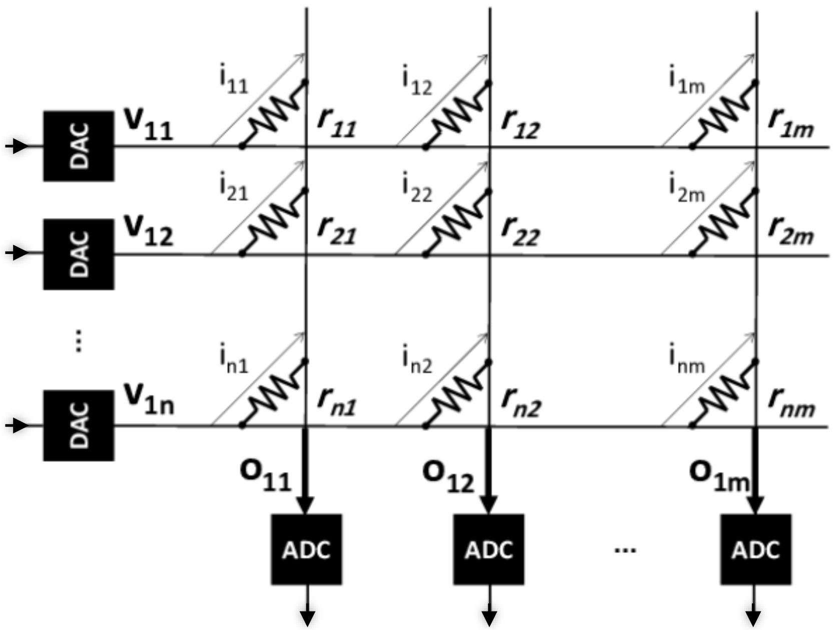
- Background & Motivation
- Crossbar-Aligned IMC Pruning
- Integer-Only IMC Quantization
- Co-optimization Framework
- Experimental evaluation
- Conclusion

Background



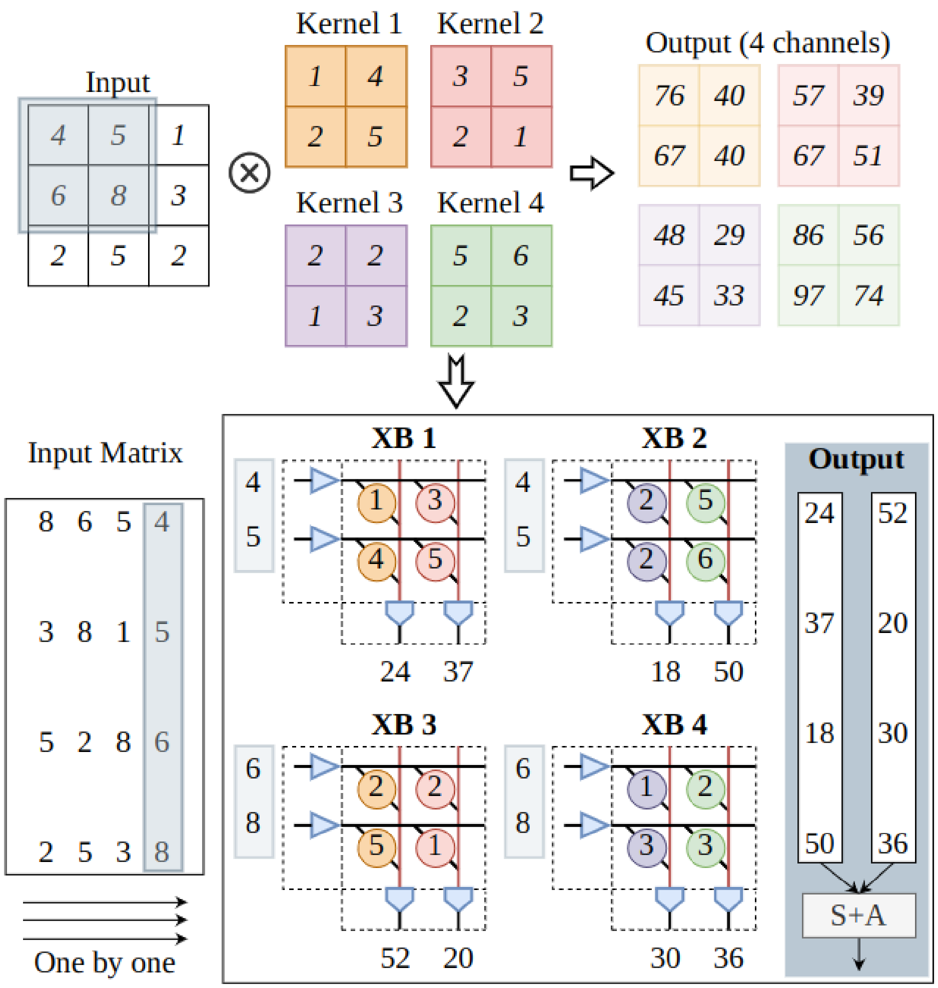
- DNNs have been adopted and implemented on low-power edge devices;
- DNNs are increasingly complex to gain more accuracy;
- DNN algorithms own a high degree of computing parallelism and extensive memory access.

Background



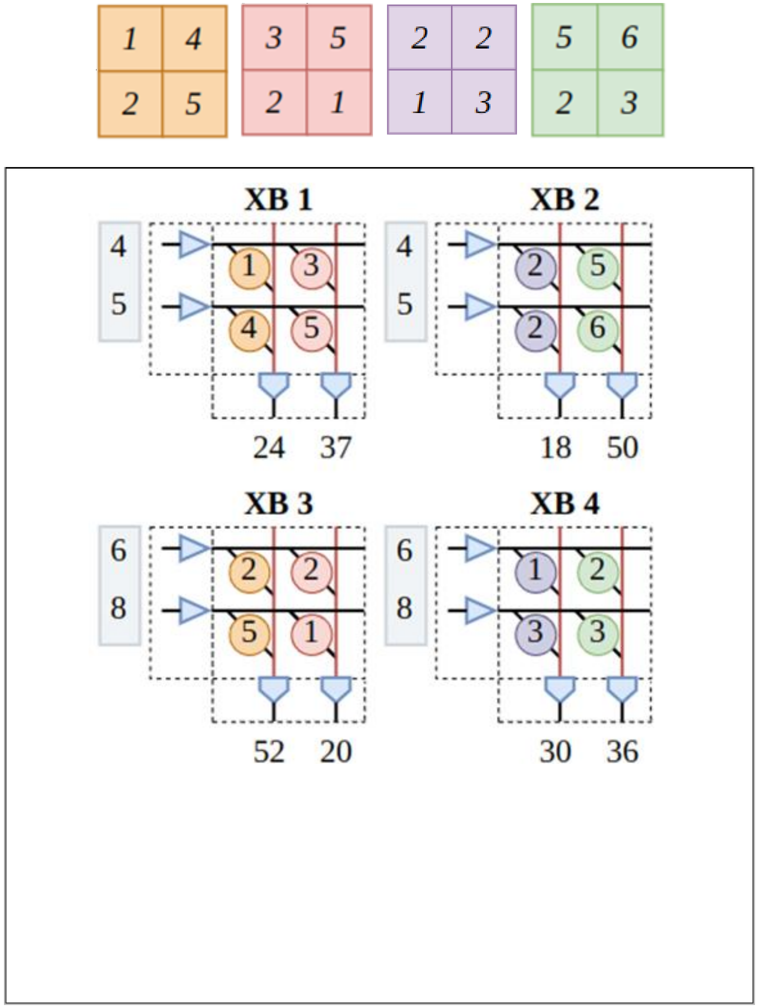
- In-memory Computing Devices perform matrix multiplication easily;
- Include many parallel arithmetic units;
- Manage computing in memory to reduce memory access;

Background



- Crossbar has large reconfigure latency and limited write endurance;
- IMC devices preload an entire model into crossbars before inference;
- It requires too many crossbars for complex models.

Motivation



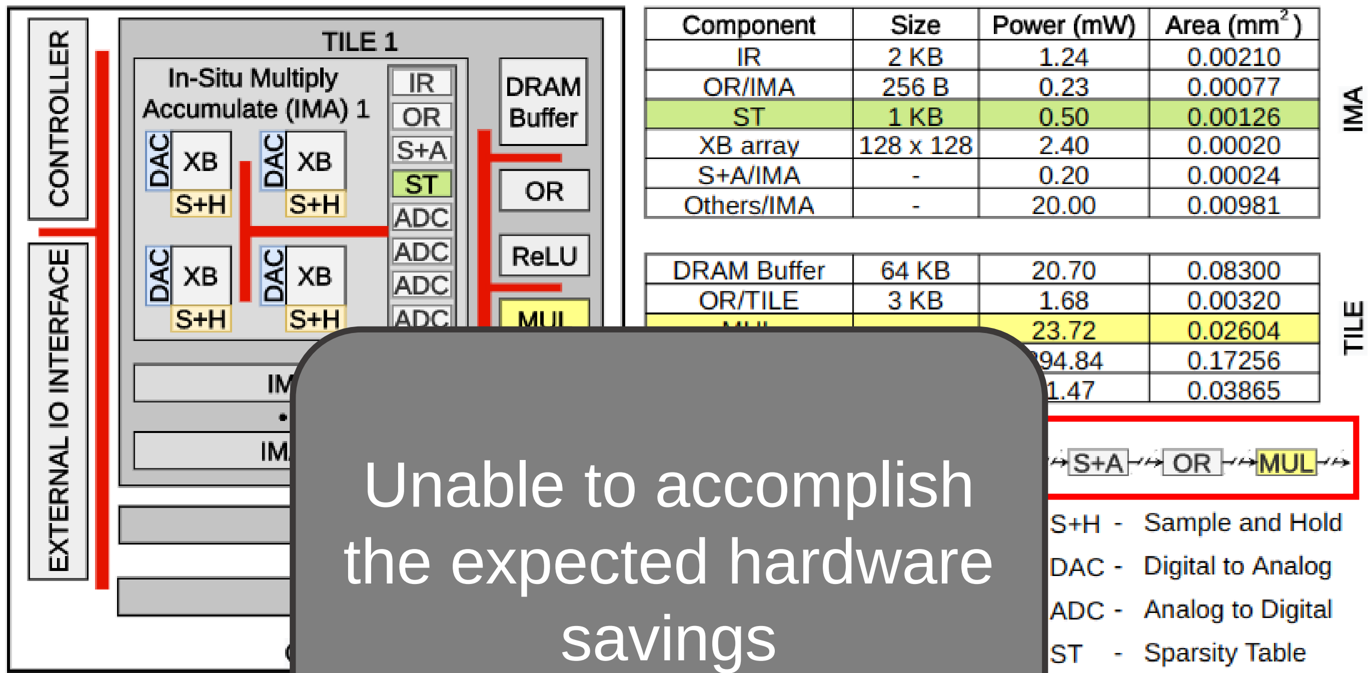
- Existing pruning methods employ fine-grained (i.e., crossbar column/row level) pruning, which trims columns/rows of weights in each crossbar;
- These fine-grained methods can reduce the number of crossbars;
- They require expensive extra hardware to align the intra-output of each crossbar.

Motivation

$$\overset{\text{Inputs/ Weights}}{\begin{bmatrix} 0.56 & 0.28 & 0.17 \\ -0.32 & 0.12 & -0.66 \\ 0.83 & -0.92 & -0.03 \end{bmatrix}} = \left\{ \overset{\text{Quantizated Values}}{\begin{bmatrix} 216 & 175 & 159 \\ 87 & 152 & 38 \\ 256 & 0 & 130 \end{bmatrix}} - \overset{\text{Zero Point}}{\boxed{134}} \right\} \times \overset{\text{Scaling Factor}}{\boxed{0.0068}}$$

- It is complicated and expensive to implement floating-point (FP) arithmetic units using IMC crossbars;
- Quantization schemes can approximate FP inputs and weights with integers;
- FP scaling factors are still employed to ensure accuracy.

Motivation



Component	Size	Power (mW)	Area (mm^2)
IR	2 KB	1.24	0.00210
OR/IMA	256 B	0.23	0.00077
ST	1 KB	0.50	0.00126
XB array	128 x 128	2.40	0.00020
S+A/IMA	-	0.20	0.00024
Others/IMA	-	20.00	0.00981
DRAM Buffer	64 KB	20.70	0.08300
OR/TILE	3 KB	1.68	0.00320
MUL	[illegible]	23.72	0.02604
[illegible]	[illegible]	[illegible]94.84	0.17256
[illegible]	[illegible]	[illegible]1.47	0.03865

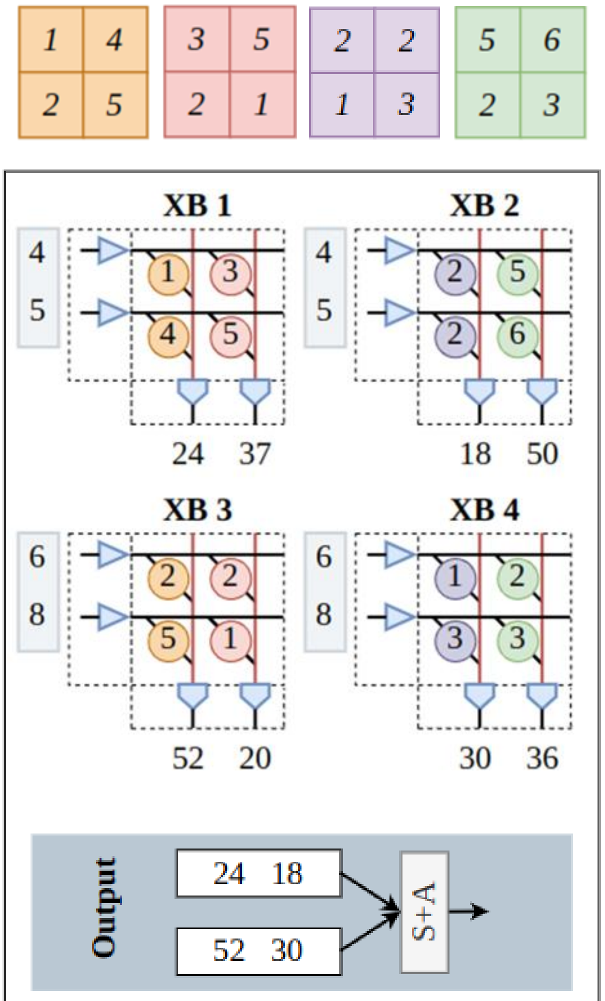
Unable to accomplish the expected hardware savings

- Extra hardwa[illegible]ning brings about 44% more storage and 10% more area;
- FP scaling factors need extra multipliers, which bring about 7% power and 9% area overhead;

Outline

- Background & Motivation
- Crossbar-Aligned IMC Pruning
- Integer-Only IMC Quantization
- Co-optimization Framework
- Experimental evaluation
- Conclusion

Crossbar-Aligned IMC Pruning



IMC-aware kernel-group pruning

- Directly remove whole kernels from DNNs;

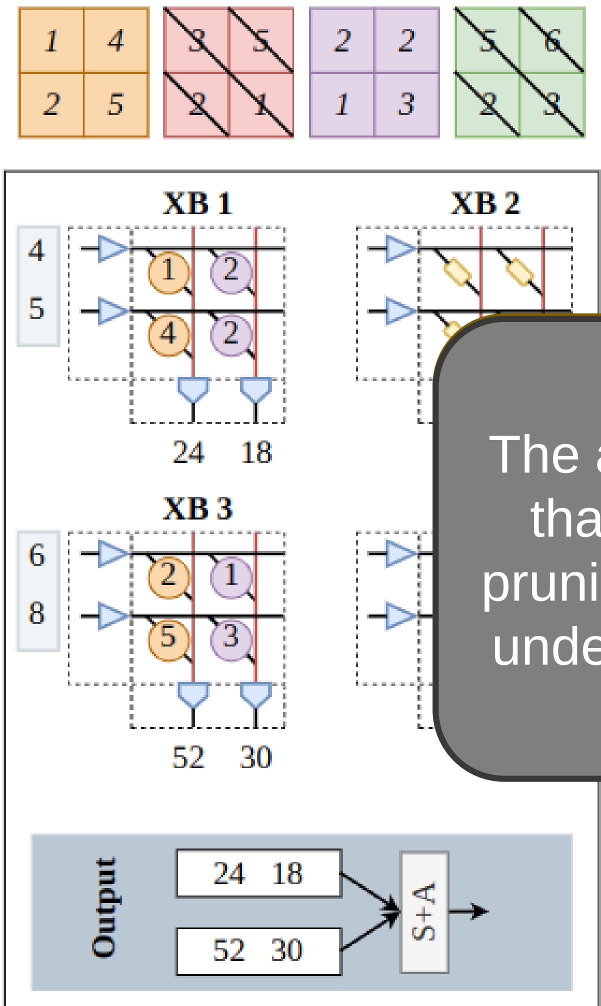
Suppose a DNN model has two connected convolutional layers (denoted as l_1, l_2).

We use $C_2 \times K_1 \times K_1 \times C_1$ and $C_3 \times K_2 \times K_2 \times C_2$ to represent the shape of kernels in these layers.

Layer l_1 needs:
$\lceil C_2/XB_w \rceil \times \lceil (K_1 \times K_1 \times C_1)/XB_h \rceil$ crossbars.

Layer l_2 needs:
$\lceil C_3/XB_w \rceil \times \lceil (K_2 \times K_2 \times C_2)/XB_h \rceil$ crossbars.

Crossbar-Aligned IMC Pruning



IMC-aware kernel-group pruning

- Directly remove whole kernels from DNNs;

the remaining kernels

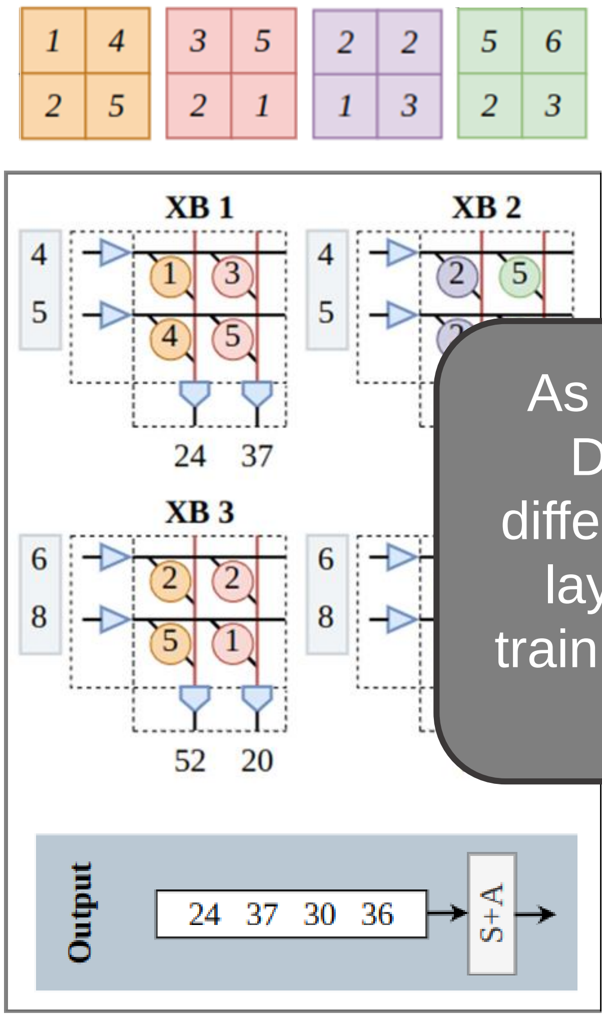
ans the reminding
ach used crossbar.

The accuracy drop is larger than that of fine-grained pruning methods, especially under a large pruning ratio.

$C_2/XB_w \times \lceil (K_1 \times K_1 \times C_1)/XB_h \rceil$ crossbars.

Layer l_2 needs:
$C_3'/XB_w \times \lceil (K_2 \times K_2 \times C_2')/XB_h \rceil$ crossbars.

Crossbar-Aligned IMC Pruning



Crossbar pruning

- Remove the crossbar-block of weights from DNNs.

al and fully-
s can be mapped
ich can compress
of layers.

mask layer after each convolutional or fully-connected layer for crossbar pruning, and each mask value is corresponding to a crossbar.

As different layers in the DNN tend to learn at different speeds, the mask layer makes the model training stage more difficult to converge.

Outline

- Background & Motivation
- Crossbar-Aligned IMC Pruning
- Integer-Only IMC Quantization
- Co-optimization Framework
- Experimental evaluation
- Conclusion

Integer-Only IMC Quantization

We first symmetrically quantize the inputs, as the voltage in IMC architectures can represent both positive and negative integers.

$$X = S_X Q_X$$

But the crossbar cells' conductivity can be only positive, so all weights and biases are asymmetrically quantized

$$W = S_W(Q_W - Z_W), B = S_B(Q_B - Z_B)$$

Take the convolutional layer as an example, the following equation depicts the quantized convolution process

$$X_1 = W \circledast X + B = S_X S_W Q_X \circledast (Q_W - Z_W) + S_B(Q_B - Z_B)$$

All values, except for Q_X, are determined during training and remain constant throughout the inference stage.

Integer-Only IMC Quantization

The training stage also determines the S_{X1} of the next layer

$$X_1 = S_{X1} Q_{X1}$$

The IMC crossbar can only perform integer arithmetic and we need to derive the Q_{X1} for the next layer's inference.

$$Q_{X1} = \frac{S_X S_W}{S_{X1}} \boxed{Q_X \circledast (Q_W - Z_W)} + \frac{S_B}{S_{X1}} \boxed{(Q_B - Z_B)}$$

Considering that IMC devices support the bit-shift operation, we can approximate scaling factors using the power of 2.

$$S_X = 2^{a1}, S_W = 2^{a2}, S_B = 2^{a3}, S_{X1} = 2^{b1}$$

Thus, the quantization only needs to reuse the S+A unit in crossbars, without overhead.

$$Q_{X1} = 2^{a1+a2-b1} Q_X \circledast (Q_W - Z_W) + 2^{a3-b1}(Q_B - Z_B)$$

Outline

- Background & Motivation
- Crossbar-Aligned IMC Pruning
- Integer-Only IMC Quantization
- Co-optimization Framework
- Experimental evaluation
- Conclusion

Co-optimization Framework

Algorithm 1: Compact DNN Learning Framework

Input: model, training data and settings, zero start epoch: s, prune_ratio: p, xb_size: x_s, quan_bits: q.

Output: the well-trained, pruned and quantized model

Function Compact_Learning($model, \delta$):
    for $e \leftarrow 1$ **to** $s - 1$ **do**
        $forward(model)$; ***#Training***
        $update\_with\_sparsity(model, \delta)$; ***#Training***
    for $e \leftarrow s$ **to** $epoch_{max}$ **do**
        $forward\_with\_quantization(model, q)$; ***#Training***
        $update\_with\_sparsity(model, \delta)$; ***#Training***
        if $e\%2 == 0$ *or* $e == epoch_{max}$ **then**
            $Imp_r \leftarrow$ Sort kernels/Crossbars by $|\delta|$;
            Find δ threshold δ^l_{th} of each layer by Imp_r, p, x_s;
            Zeroize δ^l_i if $\delta^l_i < \delta^l_{th}$; ***#Temporarily Pruning***
    Remove all weights from model with zero δ^l_i;
Initialize model and its parameters randomly;
#Phase 1 – Kernel-group pruning
Compact_Learning(model, γ);
#Phase 2 – Crossbar pruning
model_mk $\leftarrow$ Mask(model);
Compact_Learning(model_mk, mask);

We are employing "large-accuracy-loss" pruning and quantization methods to eliminate all extra hardware and achieve low-power IMC acceleration.

To minimize the accuracy loss, we propose a well-designed compact model learning framework to co-optimize our pruning and quantization schemes.

Co-optimization Framework

Algorithm 1: Compact DNN Learning Framework

Input: model, training data and settings, zero start epoch: s,
prune_ratio: p, xb_size: x_s, quan_bits: q.

Output: the well-trained, pruned and quantized model

Function `Compact_Learning`$(model, \delta)$:

- **for** $e \leftarrow 1$ **to** $s - 1$ **do**
  - $forward(model)$; ***#Training***
  - $update\_with\_sparsity(model, \delta)$; ***#Training***
- **for** $e \leftarrow s$ **to** $epoch_{max}$ **do**
  - $forward\_with\_quantization(model, q)$; ***#Training***
  - $update\_with\_sparsity(model, \delta)$; ***#Training***
  - **if** $e\%2 == 0$ *or* $e == epoch_{max}$ **then**
    - $Imp_r \leftarrow$ Sort kernels/Crossbars by $|\delta|$;
    - Find δ threshold δ^l_{th} of each layer by Imp_r, p, x_s;
    - Zeroize δ^l_i if $\delta^l_i < \delta^l_{th}$; ***#Temporarily Pruning***
- Remove all weights from model with zero δ^l_i;

Initialize model and its parameters randomly;

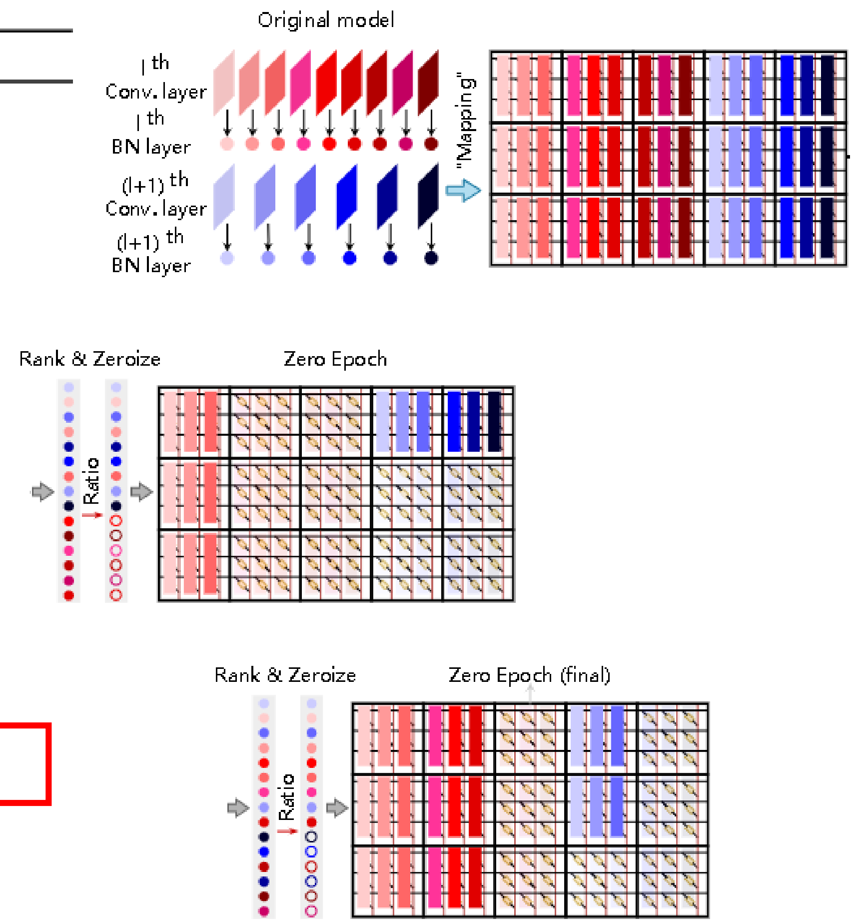
#Phase 1 – Kernel-group pruning

Compact_Learning(model, γ);

#Phase 2 – Crossbar pruning

model_mk $\leftarrow$ Mask(model);

Compact_Learning(model_mk, mask);



Co-optimization Framework

Algorithm 1: Compact DNN Learning Framework

Input: model, training data and settings, zero start epoch: s,
prune_ratio: p, xb_size: x_s, quan_bits: q.

Output: the well-trained, pruned and quantized model

Function Compact_Learning($model$, δ):
- **for** $e \leftarrow 1$ **to** $s-1$ **do**
  - $forward(model)$; ***#Training***
  - $update\_with\_sparsity(model, \delta)$; ***#Training***
- **for** $e \leftarrow s$ **to** $epoch_{max}$ **do**
  - $forward\_with\_quantization(model, q)$; ***#Training***
  - $update\_with\_sparsity(model, \delta)$; ***#Training***
  - **if** $e\%2 == 0$ *or* $e == epoch_{max}$ **then**
    - $Imp_r \leftarrow$ Sort kernels/Crossbars by $|\delta|$;
    - Find δ threshold δ^l_{th} of each layer by Imp_r, p, x_s;
    - Zeroize δ^l_i if $\delta^l_i < \delta^l_{th}$; ***#Temporarily Pruning***
- Remove all weights from model with zero δ^l_i;

Initialize model and its parameters randomly;

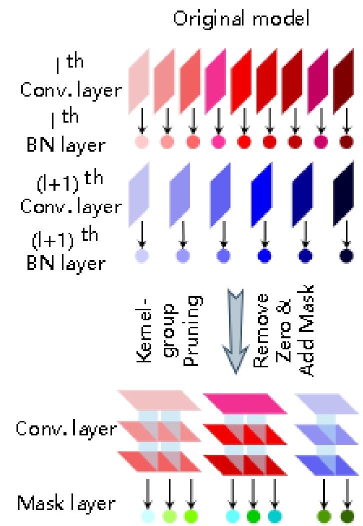
#Phase 1 – Kernel-group pruning

Compact_Learning(model, γ);

#Phase 2 – Crossbar pruning

model_mk $\leftarrow$ Mask(model);

Compact_Learning(model_mk, mask);



Co-optimization Framework

Algorithm 1: Compact DNN Learning Framework

Input: model, training data and settings, zero start epoch: s, prune_ratio: p, xb_size: x_s, quan_bits: q.

Output: the well-trained, pruned and quantized model

Function Compact_Learning($model, \delta$):
- **for** $e \leftarrow 1$ **to** $s - 1$ **do**
  - $forward(model)$; ***#Training***
  - $update\_with\_sparsity(model, \delta)$; ***#Training***
- **for** $e \leftarrow s$ **to** $epoch_{max}$ **do**
  - $forward\_with\_quantization(model, q)$; ***#Training***
  - $update\_with\_sparsity(model, \delta)$; ***#Training***
  - **if** $e\%2 == 0$ *or* $e == epoch_{max}$ **then**
    - $Imp_r \leftarrow$ Sort kernels/Crossbars by $|\delta|$;
    - Find δ threshold δ^l_{th} of each layer by Imp_r, p, x_s;
    - Zeroize δ^l_i if $\delta^l_i < \delta^l_{th}$; ***#Temporarily Pruning***
- Remove all weights from model with zero δ^l_i;

Initialize model and its parameters randomly;

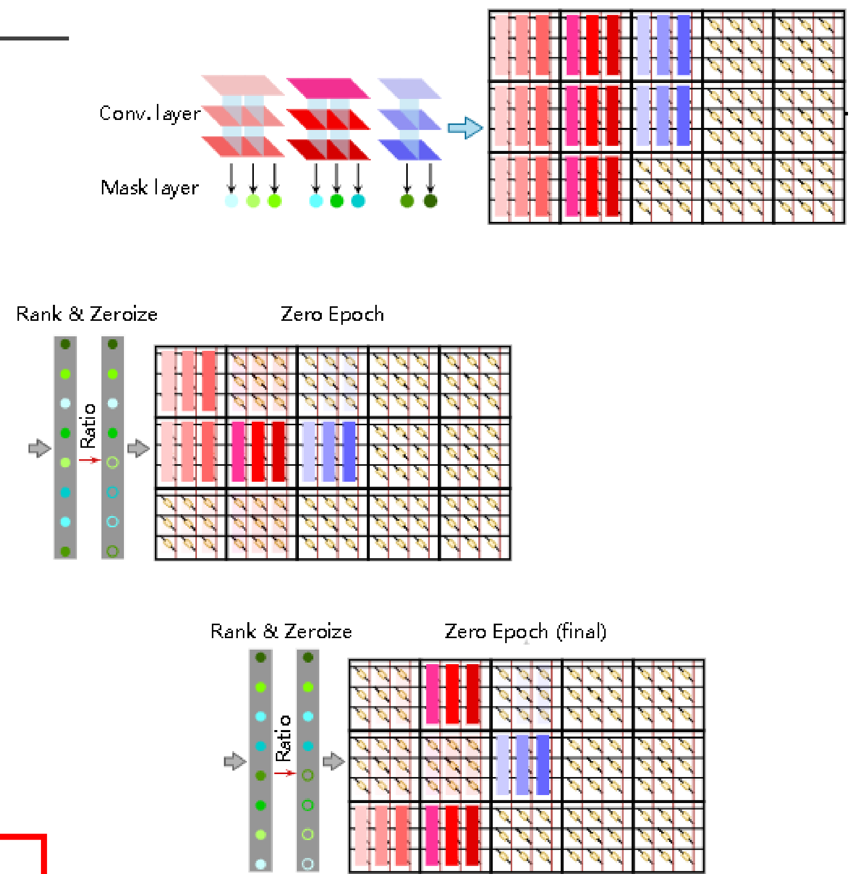
#Phase 1 – Kernel-group pruning

Compact_Learning(model, γ);

#Phase 2 – Crossbar pruning

model_mk $\leftarrow$ Mask(model);

Compact_Learning(model_mk, mask);



Co-optimization Framework

Algorithm 1: Compact DNN Learning Framework

Input: model, training data and settings, zero start epoch: s, prune_ratio: p, xb_size: x_s, quan_bits: q.
Output: the well-trained, pruned and quantized model

```
Function Compact_Learning(model, δ):
    for e ← 1 to s − 1 do
        forward(model); #Training
        update_with_sparsity(model, δ); #Training
    for e ← s to epoch_max do
        forward_with_quantization(model, q); #Training
        update_with_sparsity(model, δ); #Training
        if e%2 == 0 or e == epoch_max then
            Imp_r ← Sort kernels/Crossbars by |δ|;
            Find δ threshold δ^l_th of each layer by Imp_r, p, x_s;
            Zeroize δ^l_i if δ^l_i < δ^l_th; #Temporarily Pruning
    Remove all weights from model with zero δ^l_i;
Initialize model and its parameters randomly;
#Phase 1 – Kernel-group pruning
Compact_Learning(model, γ);
#Phase 2 – Crossbar pruning
model_mk ← Mask(model);
Compact_Learning(model_mk, mask);
```

$$X^{L+1} = \sigma^L(\delta^L \cdot (W^L \circledast X^L))$$

$$\frac{\partial loss}{\partial \delta^L} = \frac{\partial loss}{\partial X^{L+1}} \cdot \sigma^{L'} \cdot (W^L \circledast X^L)$$

$$\delta^L = 0 \Rightarrow \sigma^{L'} = 0 \Rightarrow \frac{\partial loss}{\partial \delta^L} = 0$$

$$z_{k+1} = m \cdot z_k + \frac{\partial loss}{\partial w_k}$$

$$w_{k+1} = w_k - lr \cdot z_{k+1}$$

Outline

- Background & Motivation
- Crossbar-Aligned IMC Pruning
- Integer-Only IMC Quantization
- Co-optimization Framework
- Experimental evaluation
- Conclusion

Experimental evaluation

- The IMC architecture we used is a widely-adopted ReRAM-based DNN accelerator – ISAAC;
- The crossbar size is 128 × 128, and each memristor cell stores two bits;
- We quantize DNNs to 8 bits and map each weight to 4 memristor cells;
- We use CACTI at 32nm to model the power and area of *ST* and *MUL*;
- We model the above design configurations using a modified NeuroSim simulator with the 32nm CMOS library.
- We compare our method on representative DNNs: VGG and ResNet, and on two datasets CIFAR-10 and ImageNet.

Experimental evaluation

❖ Quantization performance evaluation

Table 1: Accuracy comparison of quantization methods.

Dataset	Network	Baseline Acc.	Method	Quantized Acc.	Acc. Drop
Cifar-10	VGG-16	93.28	IAO [8]	93.14	0.14
			INT-quan	**93.39**	**-0.11**
	Resnet-56	93.34	IAO [8]	**93.44**	**-0.10**
			INT-quan	93.37	-0.03
ImageNet	Resnet-18	69.79	IAO [8]	**69.54**	**0.25**
			INT-quan	69.48	0.31

Compared to the baseline (i.e., full precision), the proposed quantization approach (denoted as INT-quan) does not result in a significant reduction of accuracy;

- This quantization approach can even provide higher accuracy than full precision in some models;
- Our technique achieves similar accuracy to the IAO quantization method, whose scaling factors are not equal to the power of 2.

Experimental evaluation

❖ Quantization performance evaluation

Table 2: Accuracy comparison of final compact models.

Network (Dataset)	Method	Baseline Acc.	Sparsity Rate	Final Acc.	Acc. Drop
VGG-16 (Cifar-10)	CSAO [12]	93.30	34.90%	93.10	0.20
	Ours	93.28	**88.25%**	**93.71**	**-0.43**
Resnet-56 (Cifar-10)	CSAO [12]	92.90	23.50%	92.50	0.40
	SPRC [15]	-	33.40%	92.80	-
	Ours	93.34	**51.36%**	**93.20**	**0.14**
Resnet-18 (ImageNet)	XBA [11]	69.31	24.89%	66.07	3.24
	SPRC [15]	69.76	26.41%	67.82	1.94
	PIM-P [2]	69.76	32.41%	68.67	1.09
	Ours	69.79	**50.50%**	**68.82**	**0.97**

All compared methods still used FP scaling factors in their quantization schemes;

All lowest accuracy reductions are marked in boldface.

- Our method can achieve a large sparsity rate on VGG-16 with even accuracy increasing (indicating VGG-16 over-fits Cifar-10 much);
- For a large dataset like ImageNet, the sparsity rate is lower than that on Cifar-10, but our method achieves a larger sparsity rate and higher accuracy.

Experimental evaluation

❖ Quantization performance evaluation

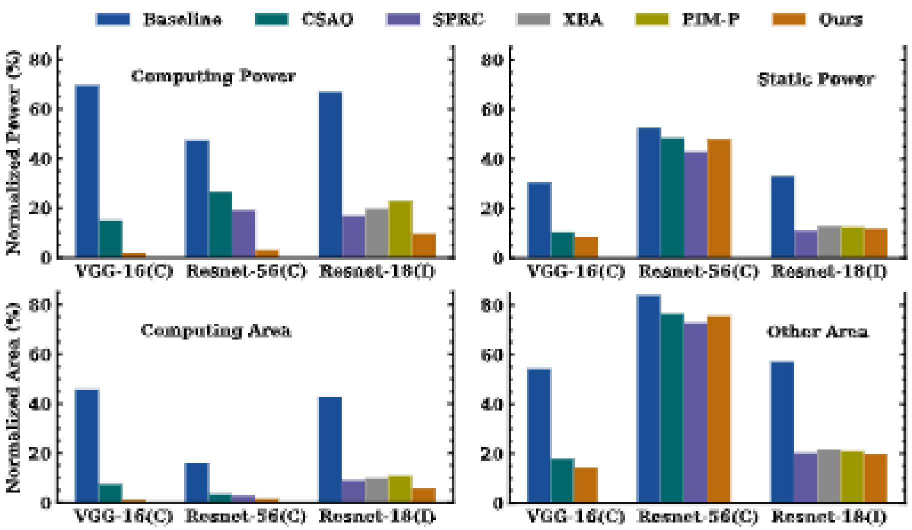


Figure 4: Evaluation of the power and area.

Crossbar-column and crossbar-row level pruning methods need the **Sparsity Table** for data alignment, and **FP** processors are needed in methods with FP scaling factors.

- The baseline is the original model, and the power and area consumption is normalized to the total consumption of the baseline.
- Our method can achieve the highest power-efficiency and area-efficiency improvement, especially for the computing parts.

Outline

- Background & Motivation
- Crossbar-Aligned IMC Pruning
- Integer-Only IMC Quantization
- Co-optimization Framework
- Experimental evaluation
- Conclusion

Conclusion

- ✓ We introduce a crossbar-aligned pruning approach to reduce crossbar usage without extra processing units for better hardware efficiency and greater integration density. It includes both kernel-group pruning and crossbar pruning to form multi-grained pruning for high accuracy and large sparsity;

- ✓ We apply a simple yet efficient integer-only quantization scheme for IMC architecture by reusing the bit-shift units. The quantization approach is co-optimized with the pruning strategy during the training process to improve accuracy;

Conclusion

- ✓ We propose a model learning framework to complete the pruning and quantization schemes. And it compacts models with a global importance rank of weights and a dynamic zero-recovery procedure, widening the exploration space for better architecture and higher accuracy.

- ✓ We demonstrate the efficiency and effectiveness of our framework with extensive experiments. Compared to state-of-the-art methods, our method can achieve a higher sparsity rate and a slighter accuracy drop without extra hardware. Thus, we can reduce computing power and computing area significantly.

Thank you

Q&A