Open Research Objects Research Objects Reviewed Results Reproduced

MSD: Mixing Signed Digit Representations for Hardware-efficient DNN Acceleration on FPGA with Heterogeneous Resources

Jiajun Wu, Jiajun Zhou, Yizhao Gao, Yuhao Ding,
Ngai Wong, Hayden Kwok-Hay So

University of Hong Kong

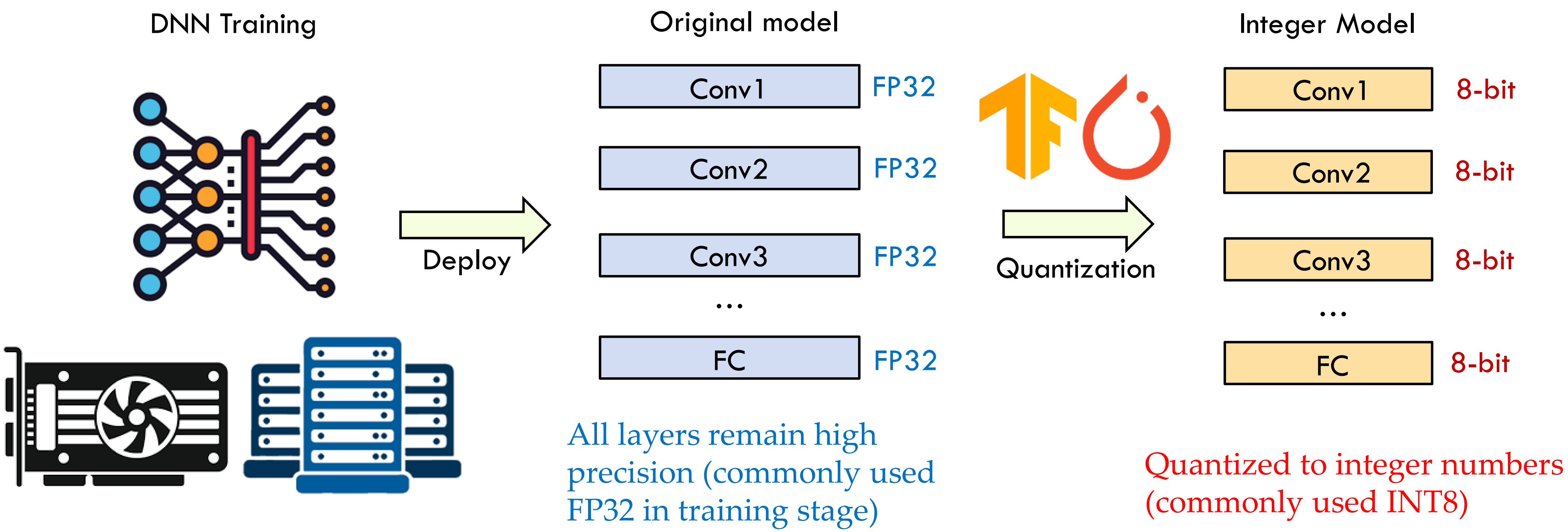
FCCM 2023

9 May 2023

Contents

- Motivation & Background
- The Proposed Methodology
- Experiment Results
- Conclusion & Future Works

DNN Quantization



DNN Training

Deploy

Original model

Conv1	FP32
Conv2	FP32
Conv3	FP32
…	
FC	FP32

All layers remain high precision (commonly used FP32 in training stage)

Quantization

Integer Model

Conv1	8-bit
Conv2	8-bit
Conv3	8-bit
…	
FC	8-bit

Quantized to integer numbers (commonly used INT8)

Can we further compress the computation of int8 to improve the inference performance without loss of accuracy?

Deploy Quantized Multiplication

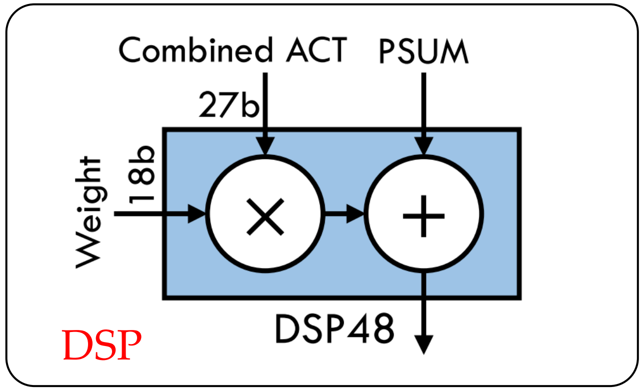
Conventional Int8 multiplication

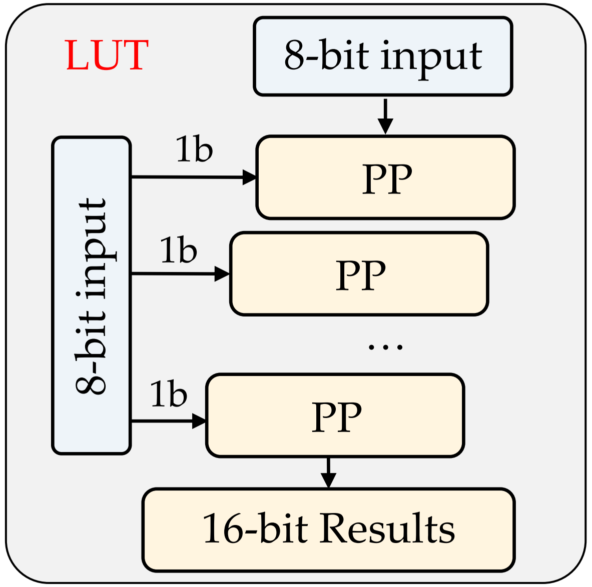
$$14 \times 27 = 00001110 \times 00011011$$

$$\begin{array}{lr}
 & 00001110 \\
 & \times\ 00011011 \\
\hline
 & 00001110 \\
\text{Partial} & 00001110 \\
\text{Products} & 00000000 \\
\text{(pp)} & 00001110 \\
 & 00001110 \\
 & 00000000 \\
 & 00000000 \\
 & +\ \ 00000000 \\
\hline
 & \text{16-bit results}
\end{array}$$

FPGA Implementation

Parallel Multiplier





Latency: 1 cycle

How do we further improve the performance?

1. To use both DSPs and LUTs for multiplication.
2. To make our LUT implementation smaller.

Bit-Serial Scheme with Bit-Sparsity

An alternative approach for the multiplier on LUT

$$14 \times 27$$
$$= 00001110 \times 00011011$$

```
  00001110
× 00011011
----------
```

Effectual bits (EB)

Partial Products (pp)

```
        00001110
       00001110
      00000000
     00001110
    00001110
   00000000
  00000000
 00000000
----------------
```

16-bit results

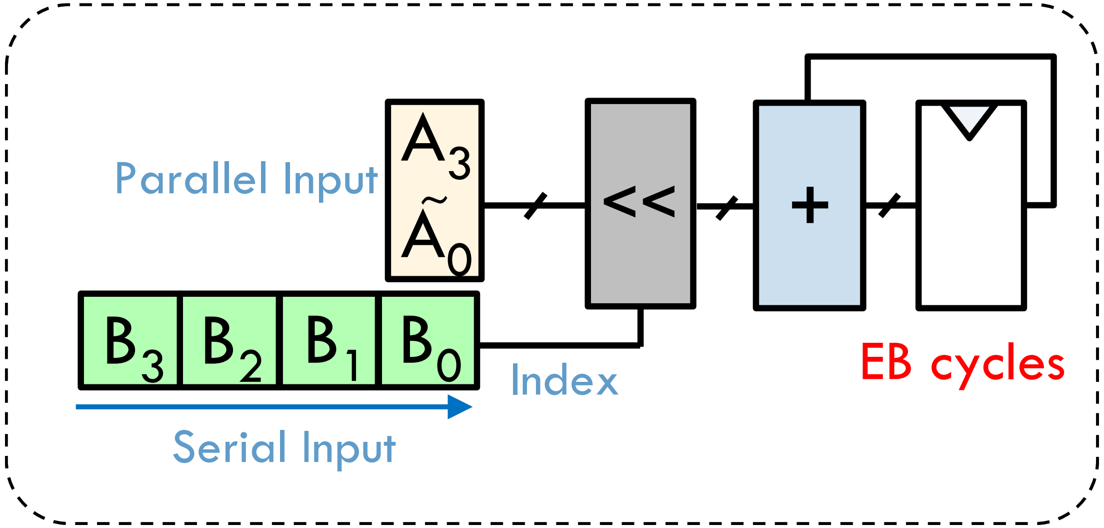
Only four partial products are effectual

Bit-serial scheme with Bit-sparsity

```
  00001110
× 00011011
----------
  00001110
 00001110
00001110
00001110
```

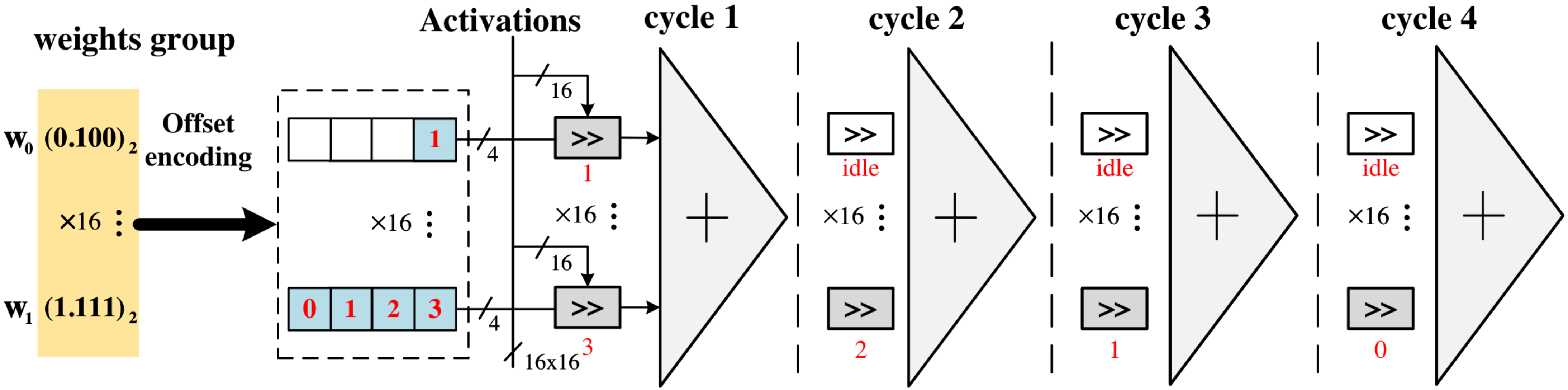
Accumulated serially

Effectual partial products



Workload Imbalance in Bit-Sparsity Scheme

The workload imbalance issue in bit-sparsity:



Ref: BitCluster (TCAD, Volume: 41, Issue: 11, November 2022)

Need a method to use a restricted and small number of EB without causing major loss of accuracy.

Contents

- Motivation & Background
- The Proposed Methodology
- Experiment Results
- Conclusion & Future Works

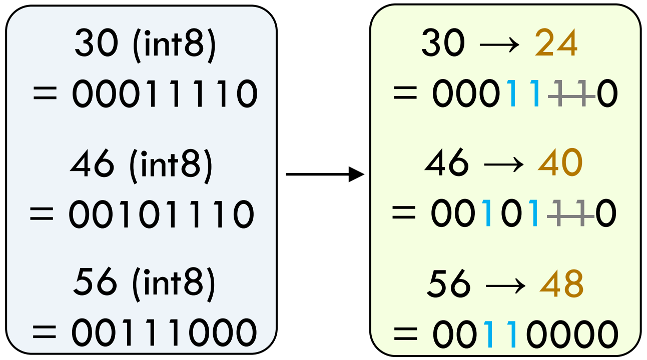
Mixing Signed-digit Representations (MSD)

- A framework that automatically partitions the DNN workloads to run on DSPs and LUTs

- We proposed a new representation called restricted signed-digit (RSD) to help restrict the number of EB

Signed-Digit Representation

To solve the imbalance issue



30 (int8)
= 00011110

46 (int8)
= 00101110

56 (int8)
= 00111000

30 → 24
= 00011~~11~~0

46 → 40
= 00101~~11~~0

56 → 48
= 00110000

Restriction: EB = 2

Large quantization error ☹

↓

Find another number format with **higher representation capability**

Our Approach: Signed-digit representation

Let $X[i]$ (i-th bit in the number with n-bit) expand to 0, 1 and -1 ($\bar{1}$):

$$X = \sum_{i=0}^{n-1} \left(X[i] \times 2^{i} \right), \qquad X[i] = \{0, \boxed{1, -1}\}$$

Effectual bits (EB)

Note that 2's complement is actually a special case of signed-digit:

$$X = (-1)^{X[n-1]} \times 2^{n-1} + \sum_{i=0}^{n-2} \left(X[i] \times 2^{i} \right), \qquad X[i] = \{0, 1\}$$

$$= \sum_{i=0}^{n-1} \left(X[i] \times 2^{i} \right), \qquad X[i] = \begin{cases} \{0, 1\} & if\ i \neq n-1 \\ \{0, -1\} & if\ i = n-1 \end{cases}$$

Hardware Does Not Change a lot!

$$\begin{aligned} &14 \times 27 \\ &= 00001110 \times 00011011 \end{aligned}$$

$$\begin{array}{r} 00001110 \\ \times\ 00011011 \\ \hline 00001110 \\ 00001110 \\ 00001110 \\ 00001110 \end{array}$$

Accumulated serially

Effectual partial products

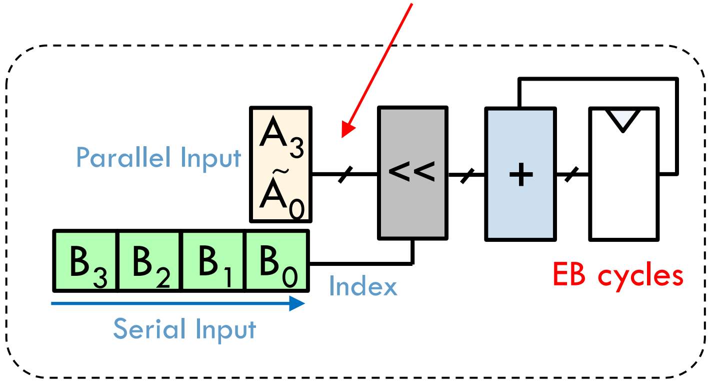
Signed-digit scheme

$$\begin{aligned} &14 \times 27 \\ &= 00001110 \times 00100\bar{1}0\bar{1} \end{aligned}$$

$$\begin{array}{r} 00001110 \\ \times\ 00100\bar{1}0\bar{1} \\ \hline -\ 00001110 \\ -\ 00001110 \\ +\,00001110 \end{array}$$

Accumulated serially

Effectual partial products

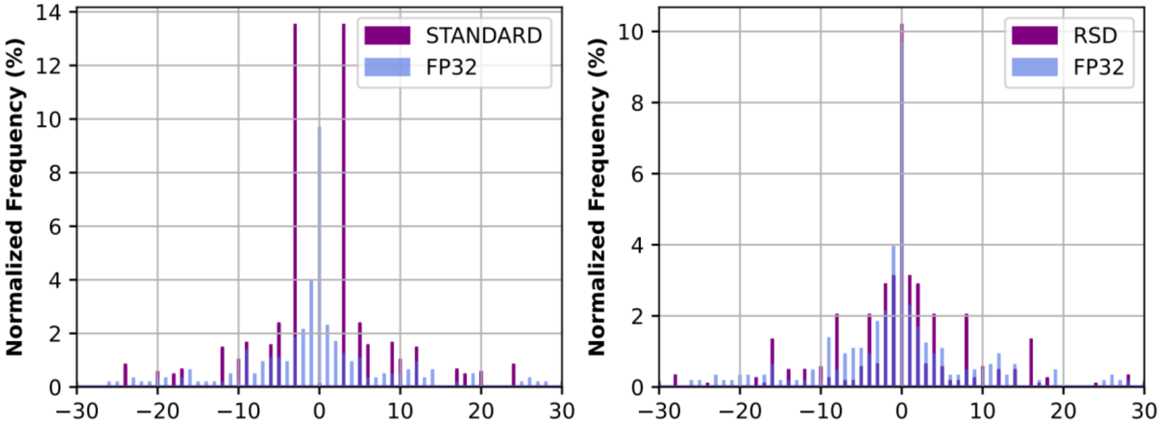


Subtraction is naturally the same with addition. We only need to add a simple circuit for negative values.

Restricted Signed-Digit

Restrict EB: Restricted signed-digit (RSD)

Original Numbers	2's complement $EB = 2$,	Error	RSD $EB = 2$	Error
30 (int8) = 00011110	30 → 24 = 00011~~11~~0	6	30 = 32 - 2 = $00100\bar{1}0$	0
46 (int8) = 00101110	46 → 40 = 00101~~11~~0	6	48 = 32 + 16 = 00110000	2
56 (int8) = 00111000	56 → 48 = 00010001	8 ☹	56 = 64 - 8 = $0100\bar{1}000$	0 ☺



RSD suits original weight distributions better

Algorithm: set up 2's integer power (0, 2, 4, 8,…) as the bases, iteratively find the base and sign bits according to the restriction. 30 -> get 32 and -2 in two iterations

Weights Encoding & Bit-serial PE Design

- We focused on layer-wise weights quantization in this work, and using the de facto 8-bit (int8) model as the starting point.

RSD

$EB = 2$

30 = 32 - 2

= $001000\bar{1}0$

48 = 32 + 16

= 00110000

16 = 32 - 16

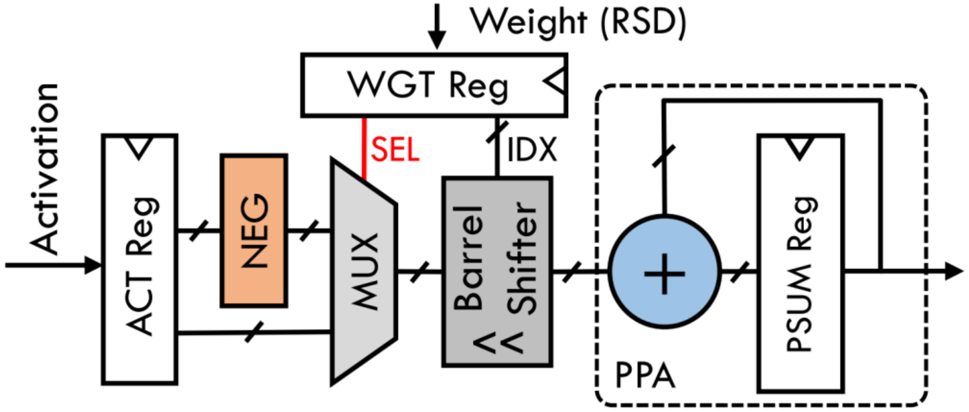
= $001\bar{1}0000$

EB's position (index) encoding →

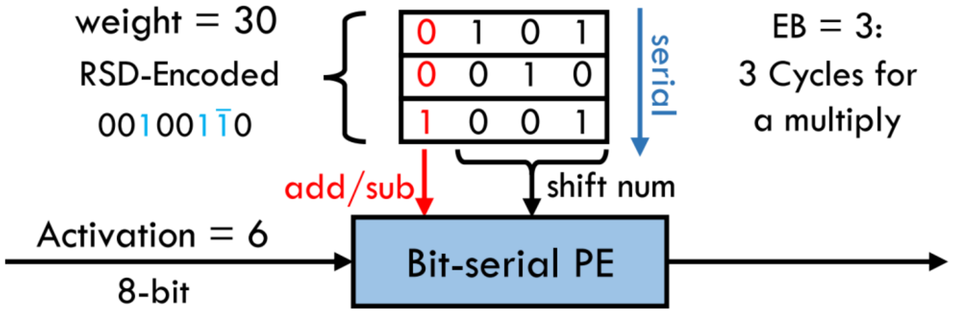
Another SEL bit to indicate {1, -1}

RSD Encode

SEL	IDX
0	1 0 1
1	0 0 1
0	1 0 1
0	1 0 0
0	1 0 1
1	1 0 0



(a) Bit-serial PE

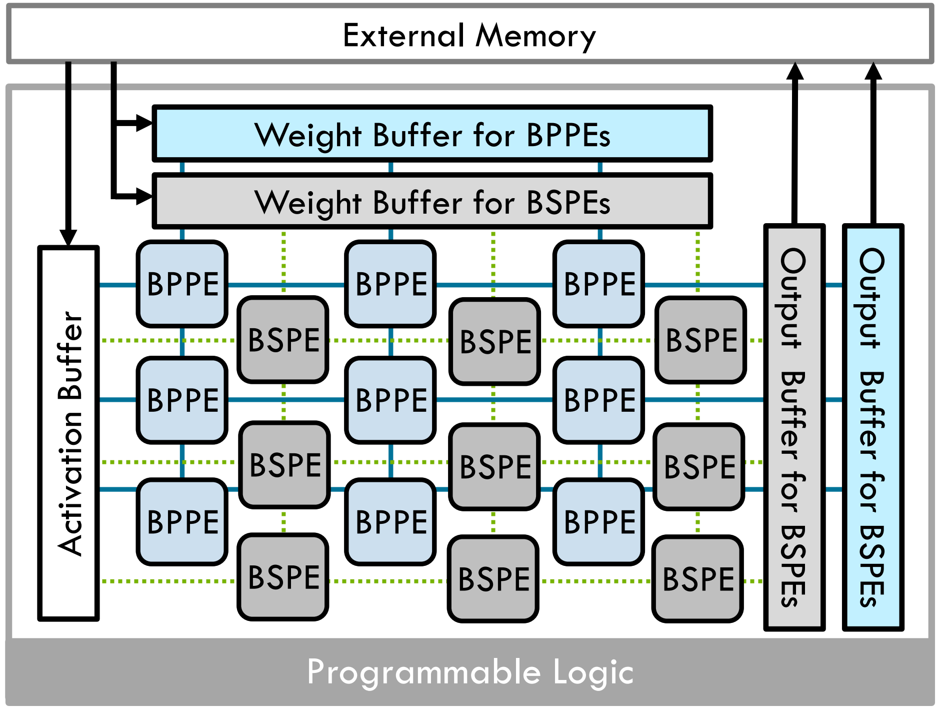


0	1	0	1
0	0	1	0
1	0	0	1

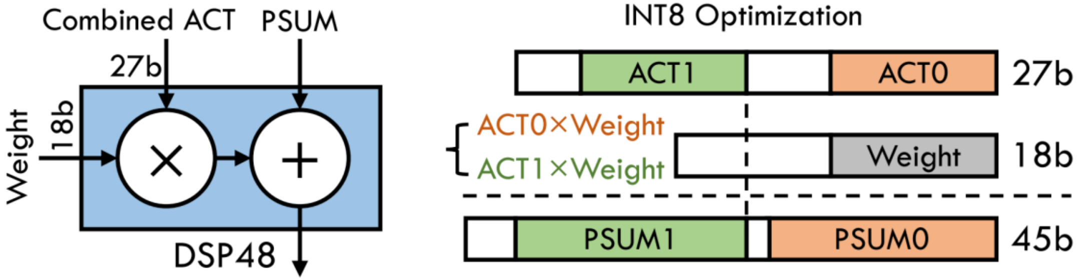
Out = 6≪5 + 6≪2 − 6≪1

(b) Bit-serial flow

Heterogeneous Architecture



- BSPE & BPPE: Bit-serial & Bit-parallel PE, set up as a systolic array
- BSPEs are implemented on LUTs, while BPPEs are based on DSPs, running simultaneously



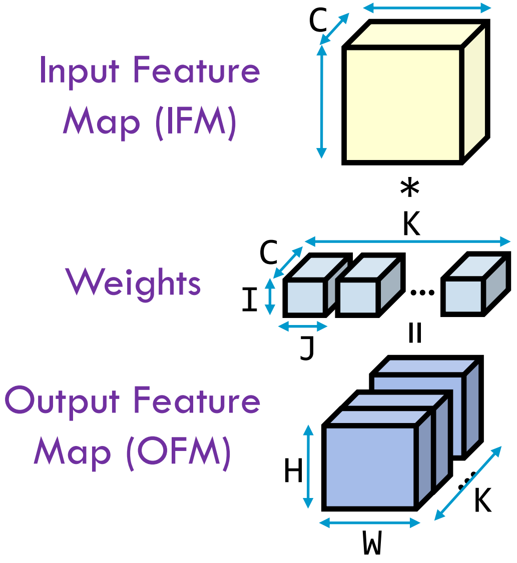
INT8 multiplication on DSPs (standard representation)

	Weights	Activations
BSPE	RSD representation	Standard
BPPE	Standard	Standard
Consistence representation by mixing signed-digit (MSD)		

- Standard 2’s complement can be regarded as a special case of signed-digit representation.

Search Schedules

- Parameters that need to be searched by the scheduler (for each layer)

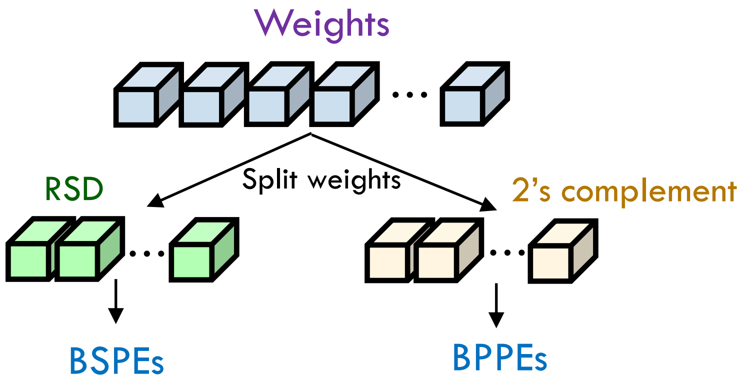


Part I: 6-dimension for-loop for each layer

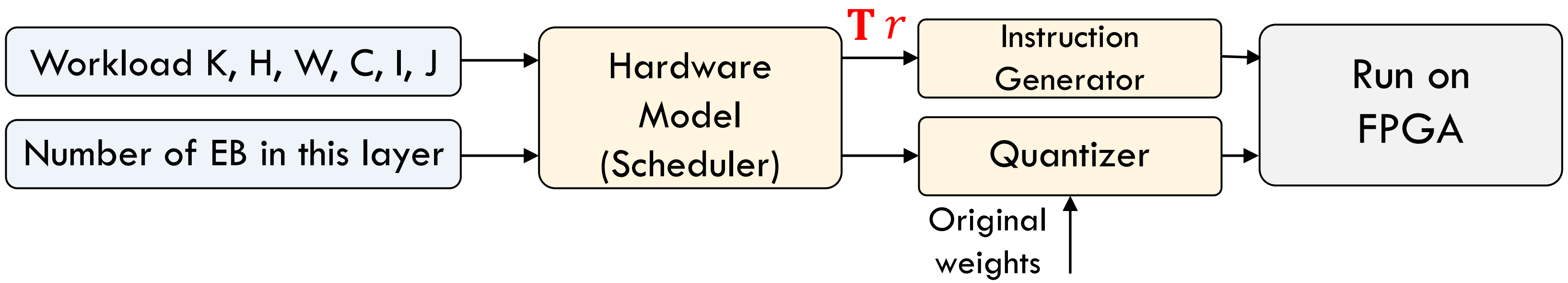
Tile size in each dimension:

$$\mathbf{T} = [t_K, t_H, t_W, t_C, t_I, t_J]$$

Part II: Weight Split Ratio (workload partitioning)



Split ratio $r = \dfrac{W_{BSPE}}{W_{Total}}$

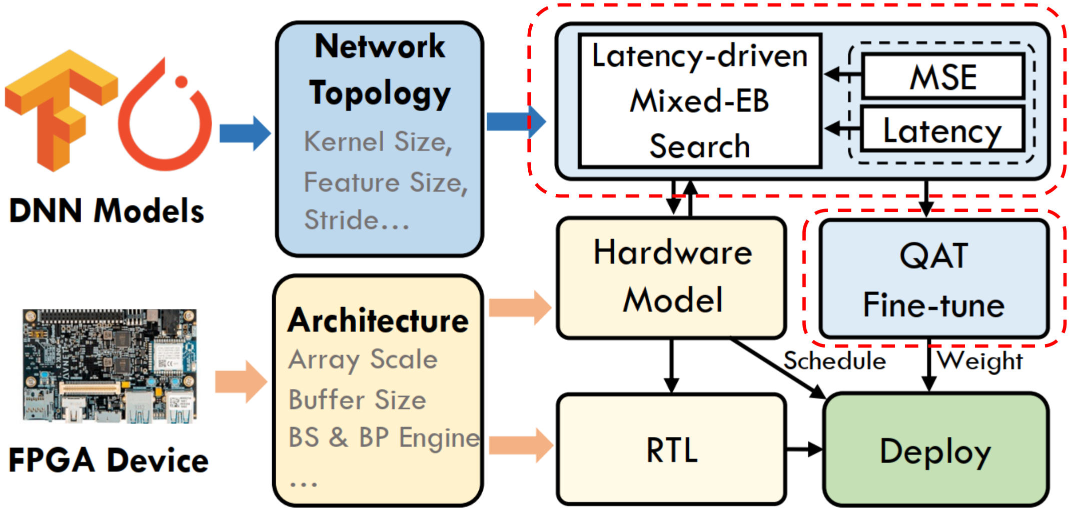

Workload K, H, W, C, I, J
Number of EB in this layer
Hardware Model (Scheduler)
T r
Instruction Generator
Quantizer
Original weights
Run on FPGA

Conv1	8-bit	EB=3
Conv2	8-bit	EB=2
…		
FC	8-bit	EB=1

Layer-wise Mixed-EB

Hardware-aware Mixed-EB Quantization

- End-to-end framework



Please Refer to PyTorch

$$\min_{\mathbf{A}} \sum_{j=1}^{m} MSE(j, \mathbf{A}[j])$$

Objective: Minimize quantization error

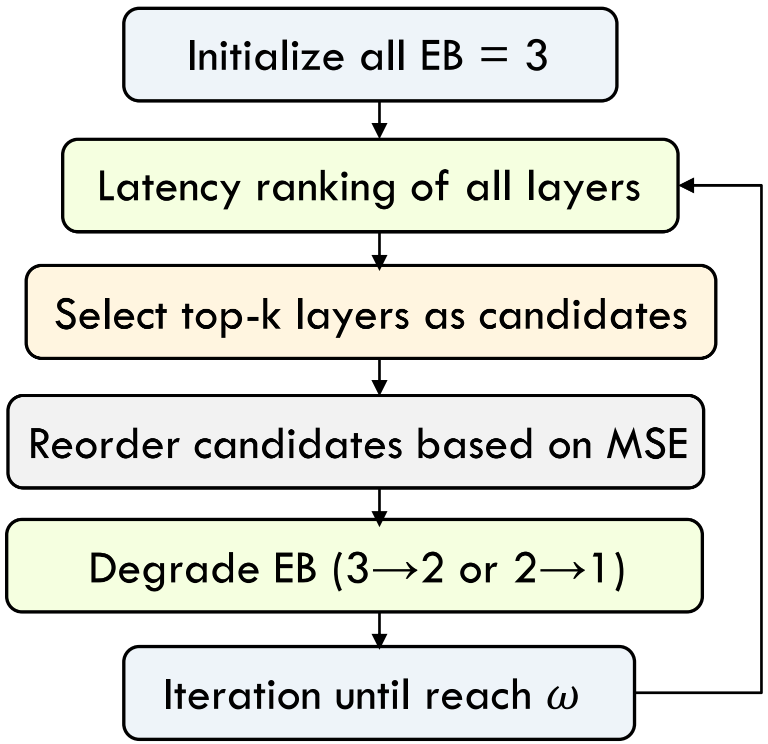
$$\text{s.t.} \quad \omega \times \sum_{j=1}^{m} Lat_{\mathrm{L}}(j, \mathbf{A}[j]) \leq Lat_{\text{base}}$$

Constraint: speedup ratio, ω

- Inputs: DNN models and FPGA device constraints
- Mixed-EB search for each layer, under the constraint of MSE and latency (calculated based on the hardware model)

$$MSE = \sqrt{\Sigma_{i=1}^{n} \left(\frac{x - \hat{x}}{\sigma_i}\right)^2}$$

Search Flow

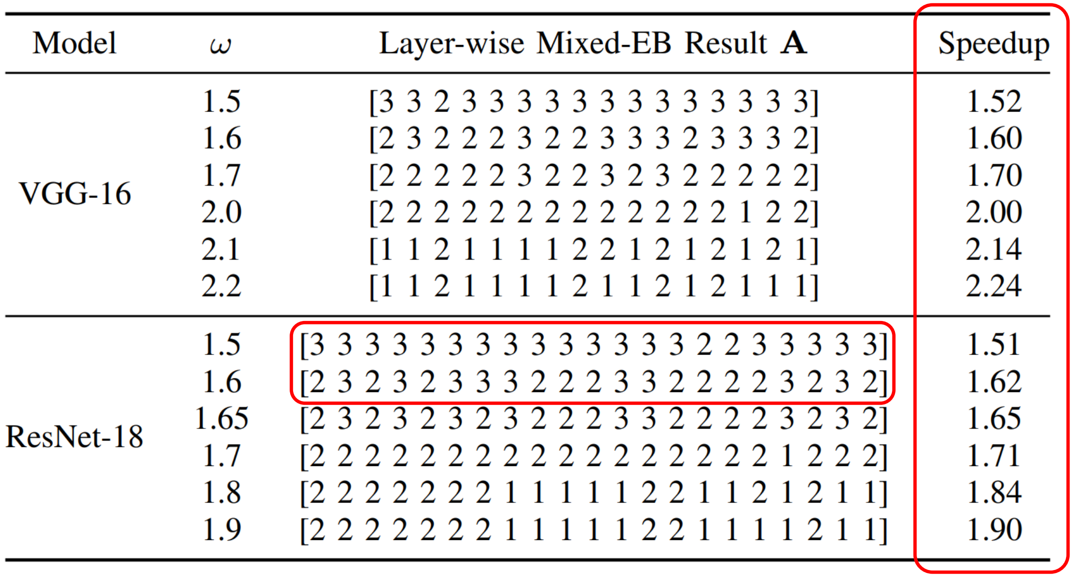

Initialize all EB = 3
Latency ranking of all layers
Select top-k layers as candidates
Reorder candidates based on MSE
Degrade EB (3→2 or 2→1)
Iteration until reach ω

Contents

- Motivation & Background
- The Proposed Methodology
- Experiment Results
- Conclusion & Future Works

Mixed-EB Quantization

TABLE I: Mixed-EB speedup results with different constraints ω. A larger ω means a higher speedup and a more aggressive quantization strategy.



Model	ω	Layer-wise Mixed-EB Result $\mathbf{A}$	Speedup
VGG-16	1.5	[3 3 2 3 3 3 3 3 3 3 3 3 3 3 3 3]	1.52
	1.6	[2 3 2 2 2 3 2 2 3 3 3 2 3 3 3 2]	1.60
	1.7	[2 2 2 2 2 3 2 2 3 2 3 2 2 2 2 2]	1.70
	2.0	[2 2 2 2 2 2 2 2 2 2 2 2 2 1 2 2]	2.00
	2.1	[1 1 2 1 1 1 1 2 2 1 2 1 2 1 2 1]	2.14
	2.2	[1 1 2 1 1 1 1 2 1 1 2 1 2 1 1 1]	2.24
ResNet-18	1.5	[3 3 3 3 3 3 3 3 3 3 3 3 3 3 2 2 3 3 3 3 3]	1.51
	1.6	[2 3 2 3 2 3 3 3 2 2 2 3 3 2 2 2 2 3 2 3 2]	1.62
	1.65	[2 3 2 3 2 3 2 3 2 2 2 3 3 2 2 2 2 3 2 3 2]	1.65
	1.7	[2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 2 2 2]	1.71
	1.8	[2 2 2 2 2 2 2 1 1 1 1 1 2 2 1 1 2 1 2 1 1]	1.84
	1.9	[2 2 2 2 2 2 2 1 1 1 1 1 2 2 1 1 1 1 2 1 1]	1.90

$$\min_{\mathbf{A}} \sum_{j=1}^{m} MSE(j, \mathbf{A}[j])$$

Objective: Minimize quantization error

$$\text{s.t.} \quad \omega \times \sum_{j=1}^{m} Lat_{\mathrm{L}}(j, \mathbf{A}[j]) \leq Lat_{\mathrm{base}}$$

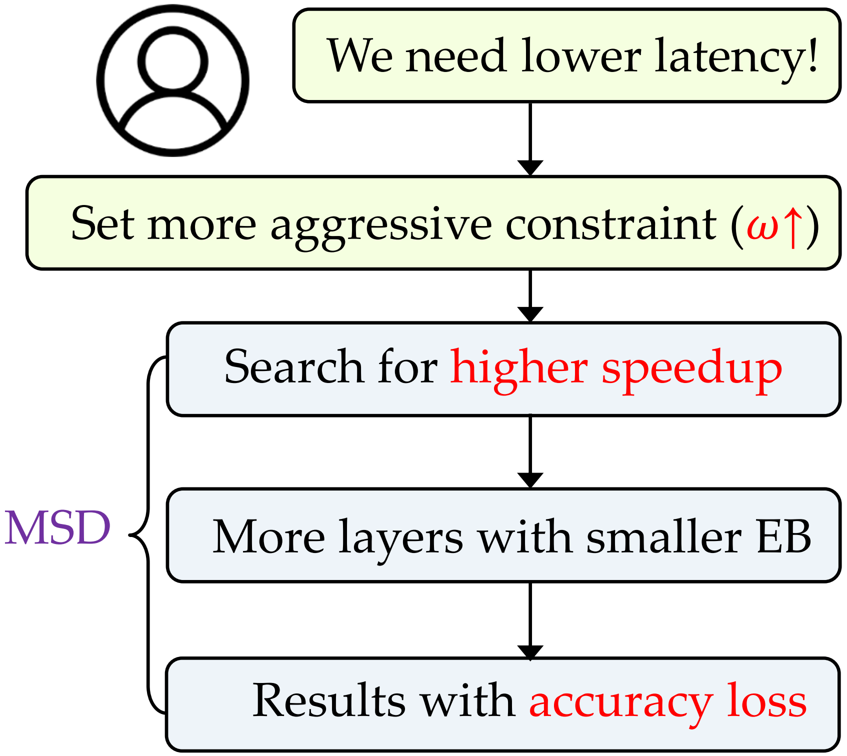
Constraint: speedup ratio, ω

The baseline is we only use DSPs for computation

- Device: Ultra-96
- Layer-wise EB configuration
- The final speedup meets the constraint of ω

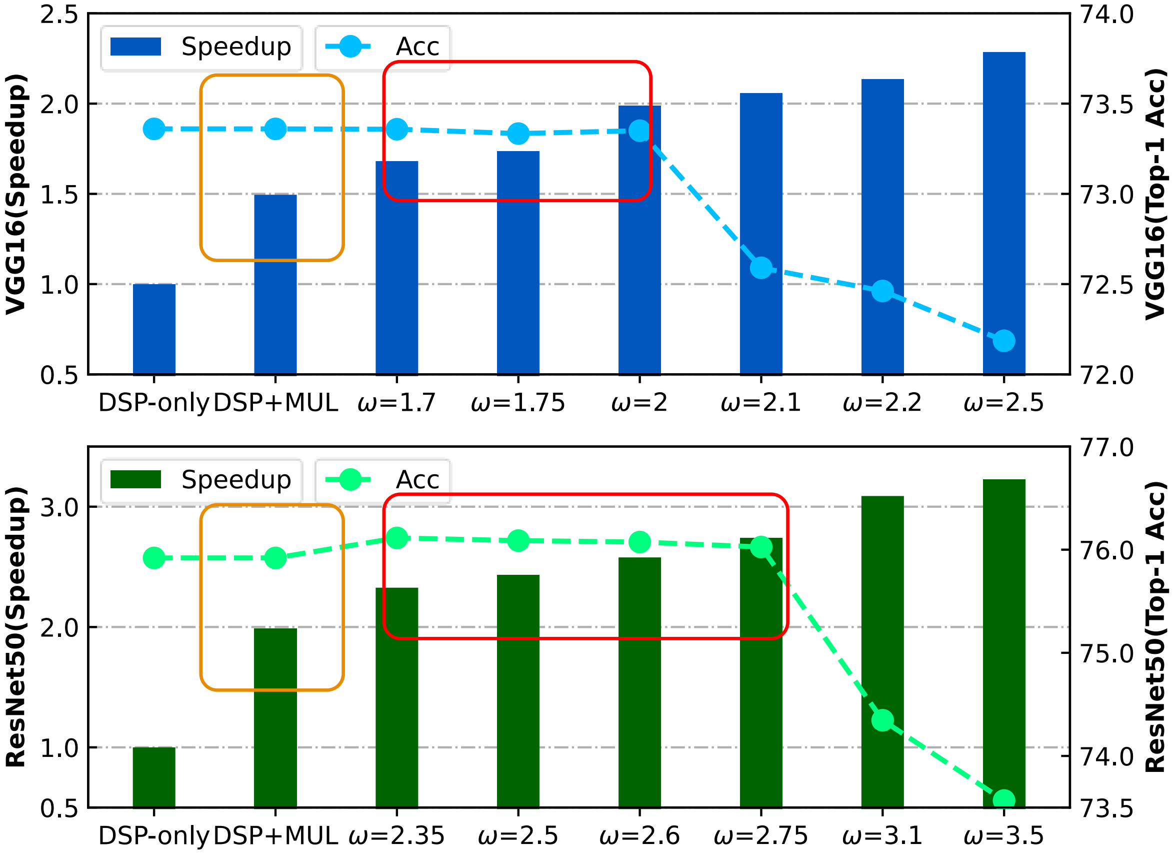
- Higher speedup constraint -> larger ω
- More aggressive search for higher speedup, to reach the constraint

Accuracy-Speedup Trade-off


We need lower latency!
Set more aggressive constraint (ω↑)
Search for higher speedup
More layers with smaller EB
Results with accuracy loss
MSD

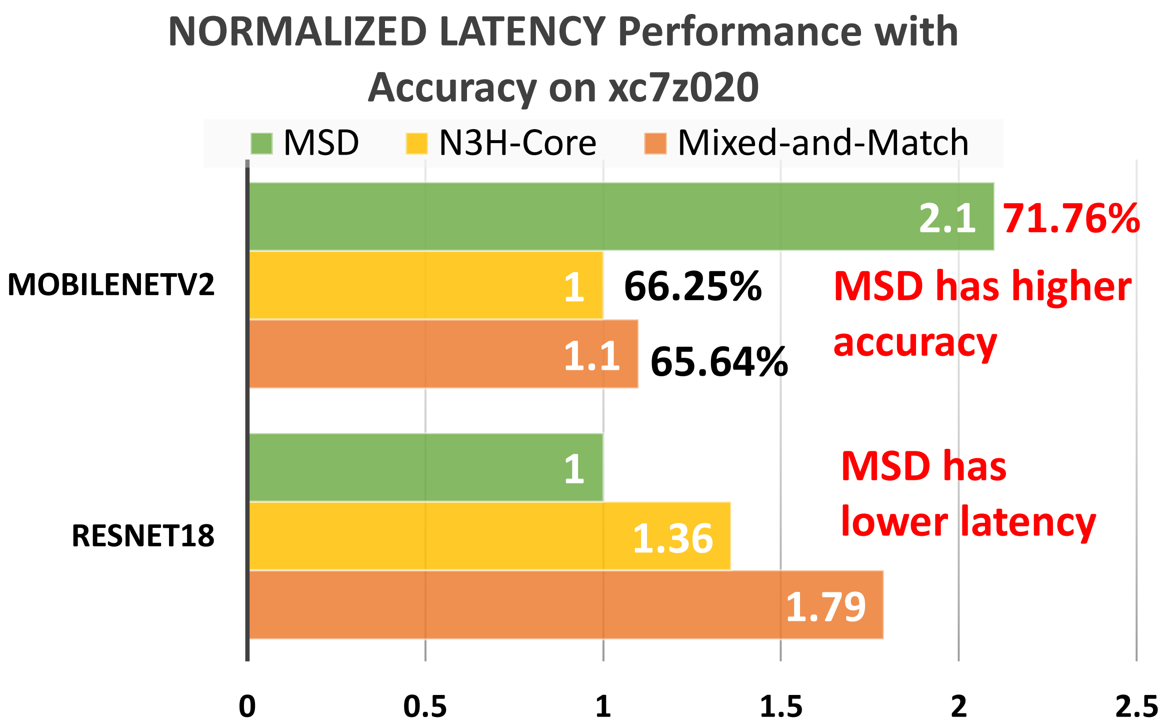
Accuracy-speedup trade-off

This area should be OK for most cases!



Also, RSD-based bit-serial scheme is more efficient than conventional parallel design on LUTs!

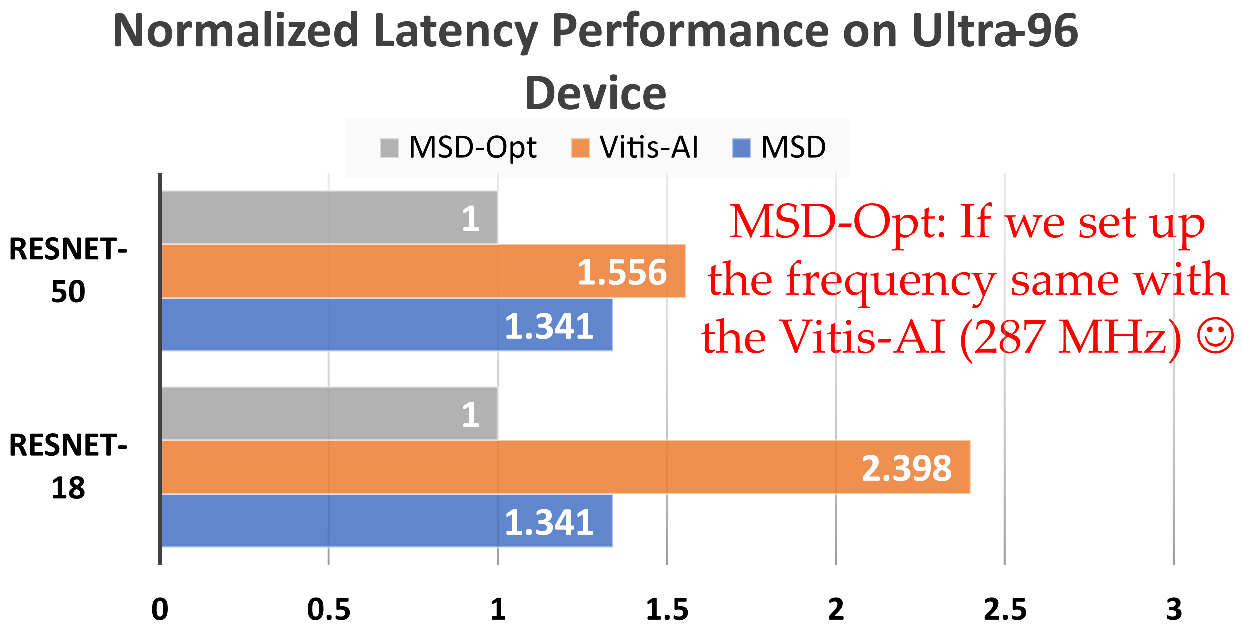
Comparison with Previous Works



NORMALIZED LATENCY Performance with Accuracy on xc7z020

MSD has higher accuracy

MSD has lower latency



Normalized Latency Performance on Ultra-96 Device

MSD-Opt: If we set up the frequency same with the Vitis-AI (287 MHz) ☺

Refer to our paper for more detailed results!

Conclusion & Future Works

- MSD framework:
  - is a heterogeneous DNN acceleration framework that utilizes both LUTs and DSPs as computation resources and to exploit bit-sparsity.
  - fine-tunes and encodes the DNN weights into a bit-sparsity-aware format, making the bit-serial computation on LUTs more efficient.
  - uses a latency-driven algorithm to search for the optimal schedule and trade-off based on layer-wise mixed EB.
- We will explore more efficient scheduling methods and exploit FPGA-layout-tailored hardware design to enhance the hardware clock frequency further.

More info of this paper in our group site!

MSD open-source in GitHub

Thanks for Your Listening

Find us in the poster session!

Q & A

More info of this paper in our group site!

MSD open-source in GitHub