ANT: Exploiting Adaptive Numerical Data Type for Low-bit Deep Neural Network Quantization

Cong Guo, Chen Zhang, Jingwen Leng, Zihan Liu,

Fan Yang, Yunxin Liu , Minyi Guo, Yuhao Zhu

饮 水 思 源 • 爱 国 荣 校

01

Background & Motivation

02

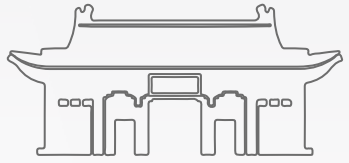
ANT Design

03

Integration

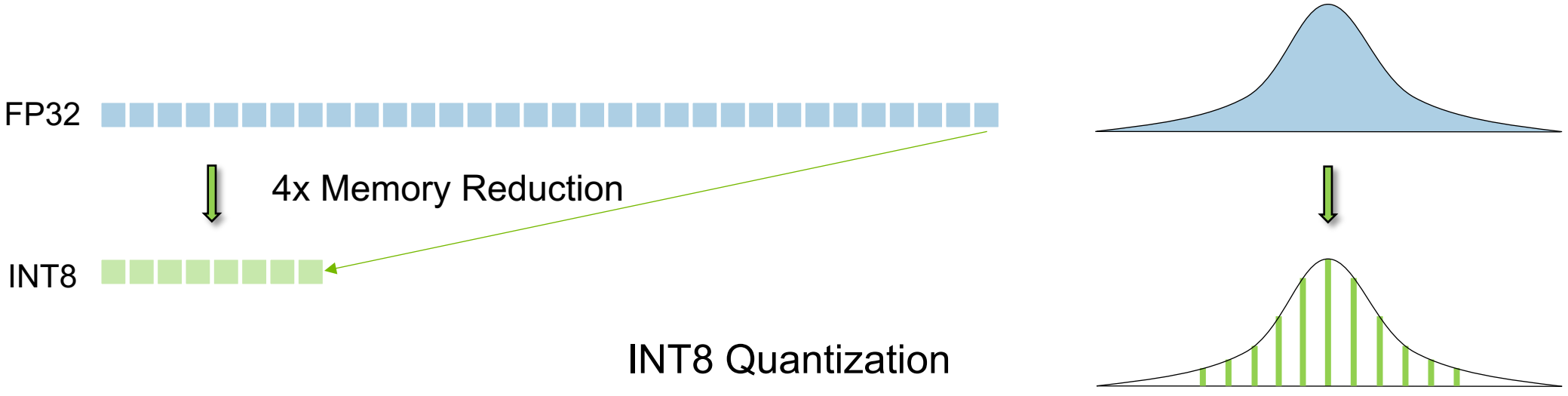
04

Evaluation



Quantization is Key to DNN Acceleration

- Quantization can effectively reduce the parameters and computation of DNN.

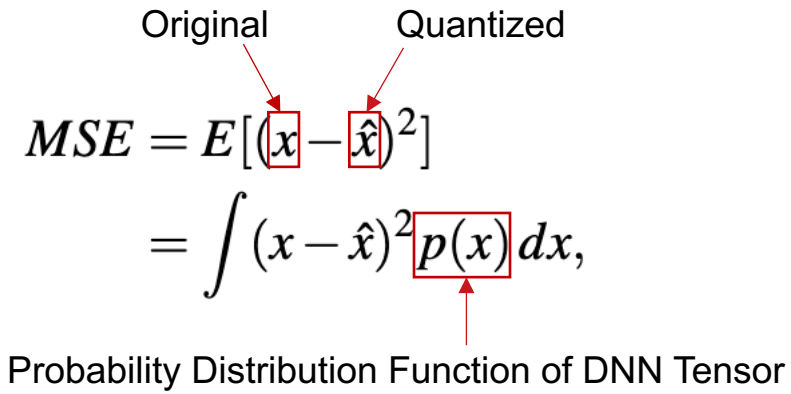


- Efficient memory and computation

Quantization and Tensor Distribution

- To achieve a better accuracy, MSE is widely used to measure quantization error.

MSE (Mean Square Error)



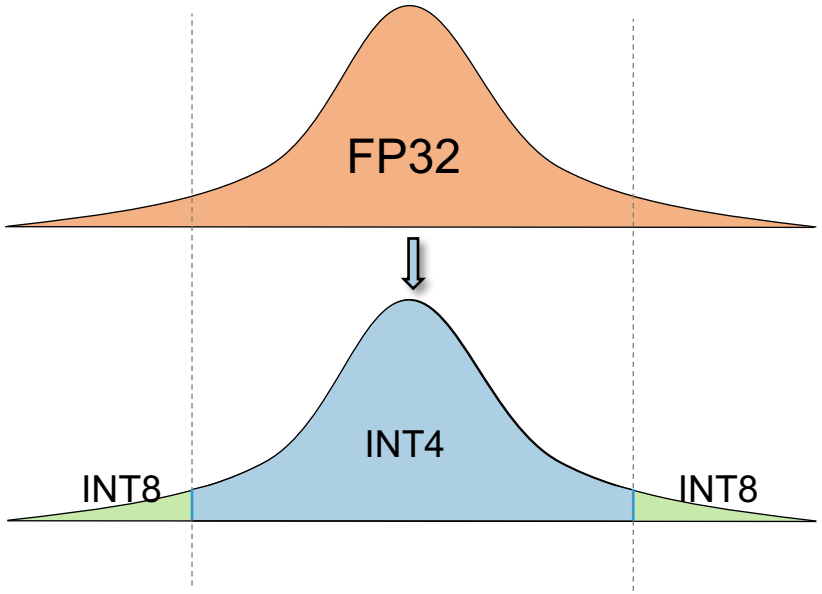
The MSE has a strong correlation with the tensor distribution.

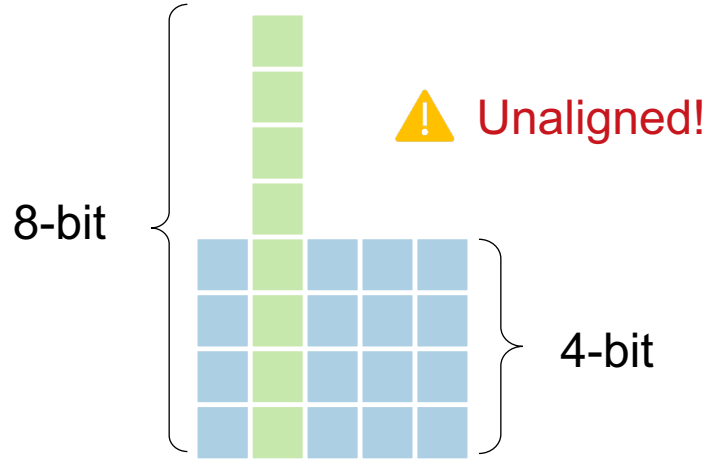
- New quantization architectures need to be aware of the tensor distribution.

[OMSE, ICCV19], [LSQ+, CVPR20], [Ada Round, ICLR20]

Prior Quantization Architecture

Outlier-aware Quantization





Quantize a tensor using two kinds of precision

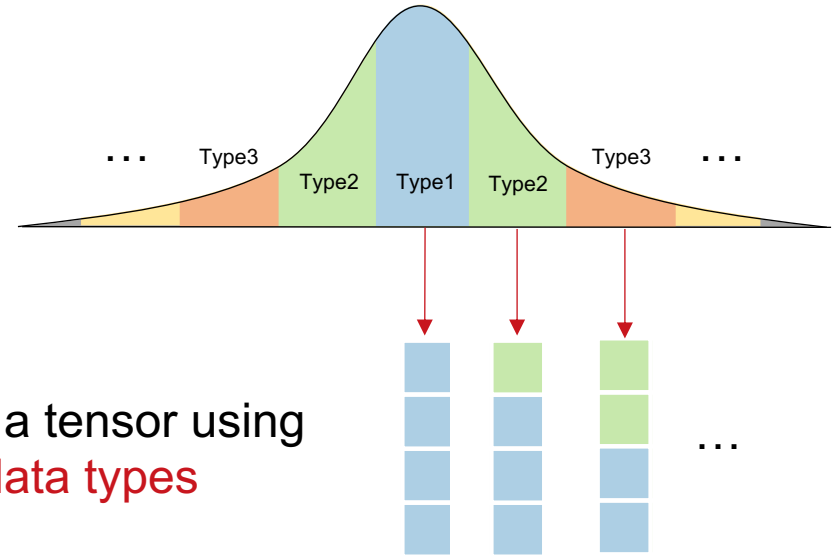
Significant hardware overhead

- Prior quantization architectures adopt variable-length data types to fit the distribution, leading to unaligned memory access.

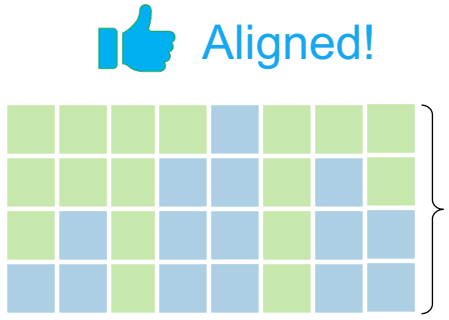
[OLAccel, ISCA18], [GOBO, Micro20], [DRQ, ISCA20]

Adaptive Numerical Data Type (ANT)

- ANT uses fixed-length data types to be hardware-friendly.
- ANT uses multiple data types to fit the distribution.
- The tensor memory is aligned and memory access is efficient.



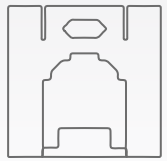
Quantize a tensor using multiple data types



- ANT can achieve low bit and low overhead at the same time.

01

Background
&
Motivation



02

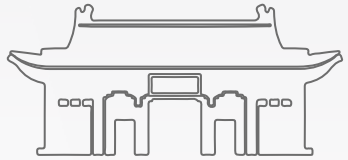
ANT Design

03

Integration

04

Evaluation



Key Insights in ANT

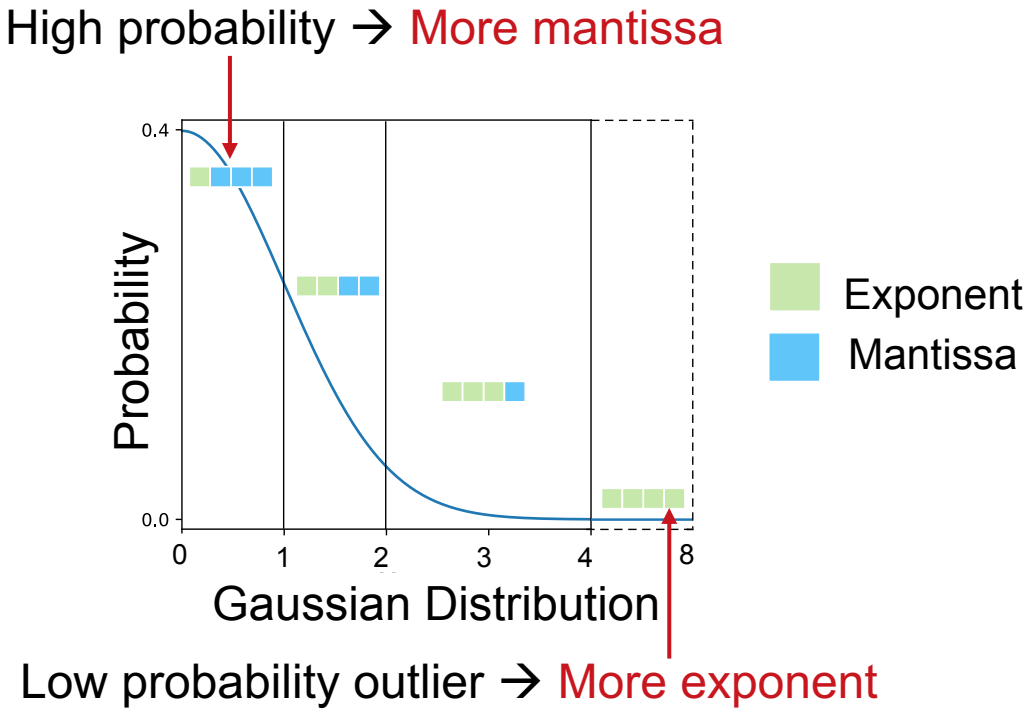
- Fixed-length Data Type
  - Hardware-friendly
- Intra-tensor Adaptivity
  - Adapt to the importance and probability of values.
- Inter-tensor Adaptivity
  - Adapt to the various distributions.

Intra-tensor ANT

For the normal distribution, we divide the elements into multiple intervals. For each interval, we use different bit lengths for exponent and mantissa to adapt to the importance (probability) of values.

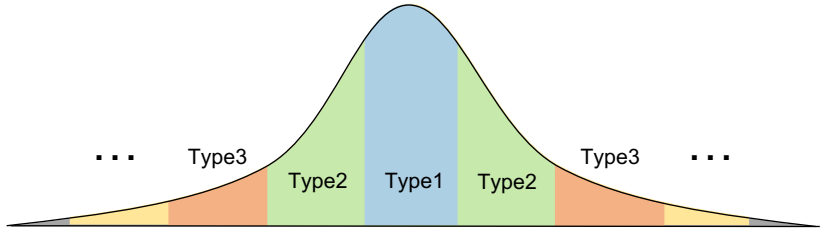
But the full data type is fixed-length.

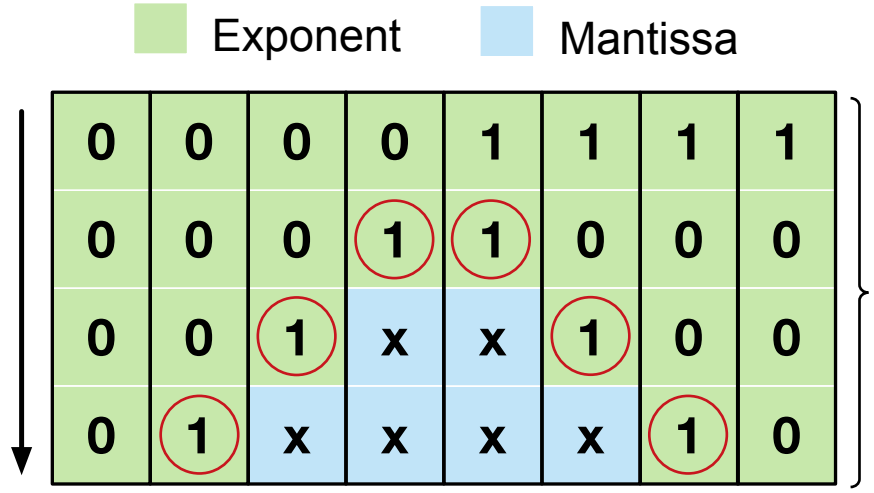
We call the proposed adaptive data type Flint, which can combine the advantages of float and int.



Intra-tensor ANT: That can reduce the MSE for quantized tensor.

Intra-tensor ANT: Flint





4-bit Unsigned Flint Encoding

To mark the boundary between the exponent and mantissa field, we use the first appearance of bit '1' after the most significant bit. We call this first-one encoding.

Flint Encoding Design

Intra-tensor ANT: Flint

The 4-bit unsigned Flint decoding function

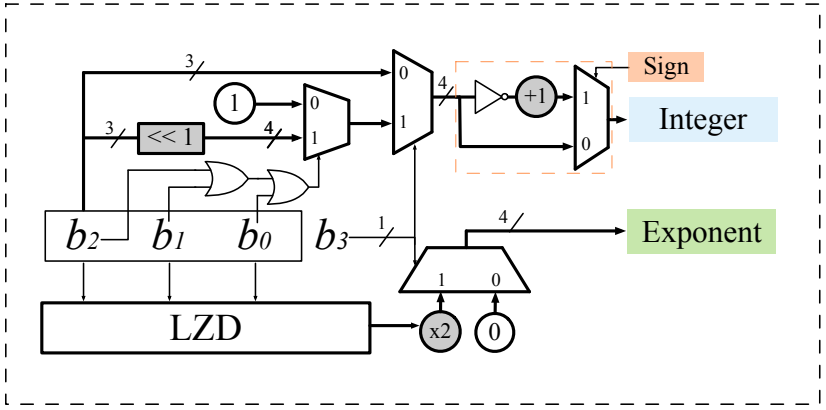
$$\text{Base Integer} = \begin{cases} b_2b_1b_0 & , \quad b_3 = 0 \\ b_2b_1b_0 << 1 & , \quad b_3 = 1 \\ 1 & , \quad x \quad = 1000_2 \end{cases}$$

$$\text{Exponent} = \begin{cases} 0 & , \quad b_3 = 0 \\ 2 \times \texttt{LZD}(b_2b_1b_0) & , \quad b_3 = 1 \end{cases}$$

Leading Zero Detector

RTL Verilog

The 4-bit Flint decoder



The LZD can detect the zero number before the first ‘1’, i.e., the pattern of first-one encoding.

Flint Decoding Design

Intra-tensor ANT: Flint

The 4-bit unsigned Flint decoding function

$$\text{Base Integer} = \begin{cases} b_2b_1b_0 & , \quad b_3 = 0 \\ b_2b_1b_0 << 1 & , \quad b_3 = 1 \\ 1 & , \quad x = 1000_2 \end{cases}$$

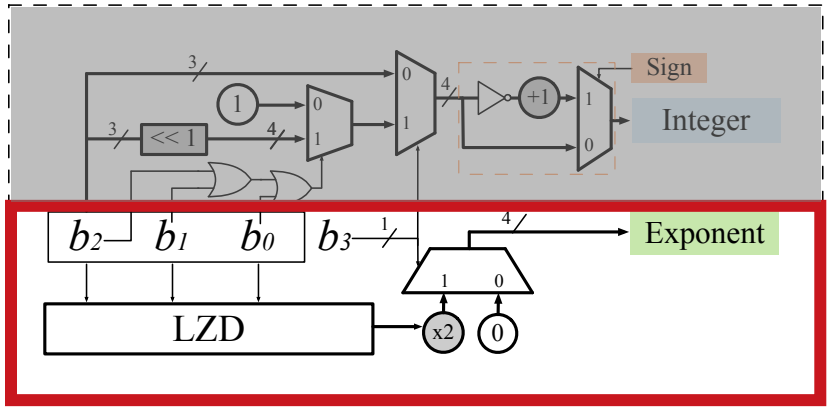
$$\text{Exponent} = \begin{cases} 0 & , \quad b_3 = 0 \\ 2 \times \text{LZD}(b_2b_1b_0) & , \quad b_3 = 1 \end{cases}$$

Leading Zero Detector

The LZD can detect the zero number before the first '1', i.e., the pattern of first-one encoding.

RTL Verilog

The 4-bit Flint decoder



1. Use LZD to decode the first-one encoded exponent.

Flint Decoding Design

Intra-tensor ANT: Flint

The 4-bit unsigned Flint decoding function

$$\text{Base Integer} = \begin{cases} b_2b_1b_0 & , \quad b_3 = 0 \\ b_2b_1b_0 << 1 & , \quad b_3 = 1 \\ 1 & , \quad x \quad = 1000_2 \end{cases}$$

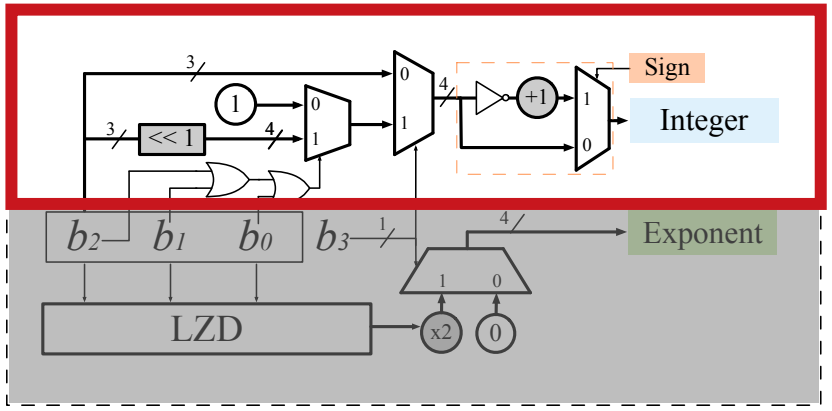
$$\text{Exponent} = \begin{cases} 0 & , \quad b_3 = 0 \\ 2 \times \mathtt{LZD}(b_2b_1b_0) & , \quad b_3 = 1 \end{cases}$$

Leading Zero Detector

The LZD can detect the zero number before the first '1', i.e., the pattern of first-one encoding.

RTL Verilog

The 4-bit Flint decoder



1. Use LZD to decode the first-one encoded exponent.

2. Use a shifter to decode the integer.

Flint Decoding Design

Intra-tensor ANT: Flint

The 4-bit unsigned Flint decoding function

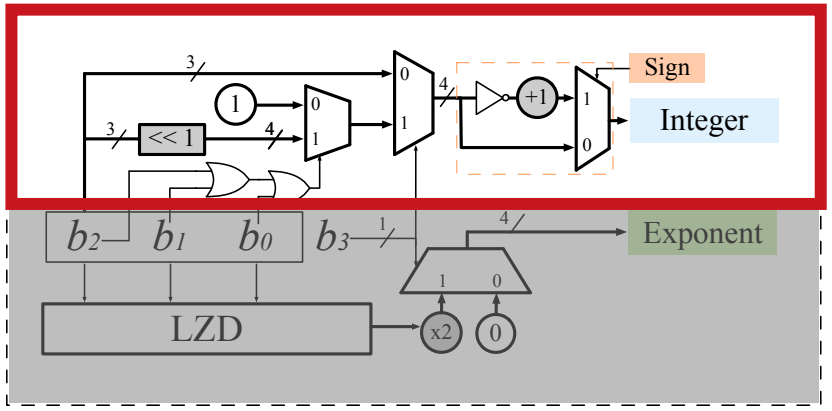
$$\text{Base Integer} = \begin{cases} b_2b_1b_0 & , \quad b_3 = 0 \\ b_2b_1b_0 << 1 & , \quad b_3 = 1 \\ 1 & , \quad x \quad = 1000_2 \end{cases}$$

$$\text{Exponent} = \begin{cases} 0 & , \quad b_3 = 0 \\ 2 \times \mathtt{LZD}(b_2b_1b_0) & , \quad b_3 = 1 \end{cases}$$

Leading Zero Detector

RTL Verilog

The 4-bit Flint decoder



The LZD can detect the zero number before the first '1', i.e., the pattern of first-one encoding.

The decoder coverts the Flint into an exponent and integer.

1. Use LZD to decode the first-one encoded exponent.

2. Use a shifter to decode the integer.

Flint Decoding Design

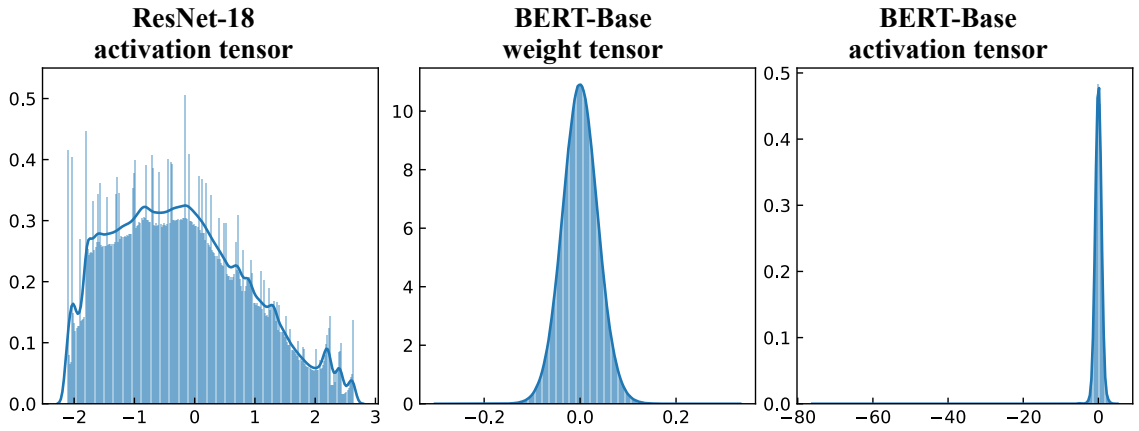
Inter-tensor ANT

We design a selection algorithm for Inter-tensor ANT based on MSE.

We select the data type with the minimum MSE to quantize each tensor.

$$MSE = E[(x - \hat{x})^2]$$
$$= \int (x - \hat{x})^2 p(x)\, dx,$$

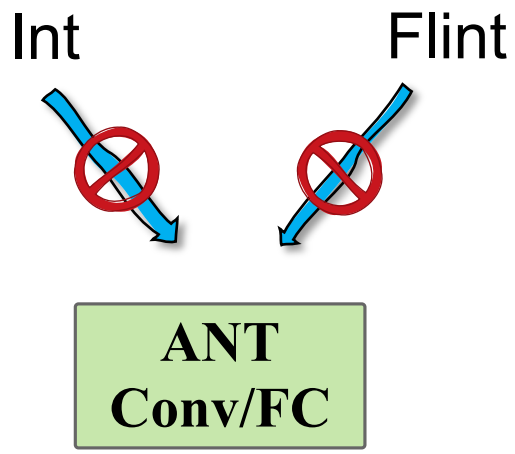
Adapt to the various distributions



Inter-tensor ANT: Tensor-wise Adaptive Numeric Data Type

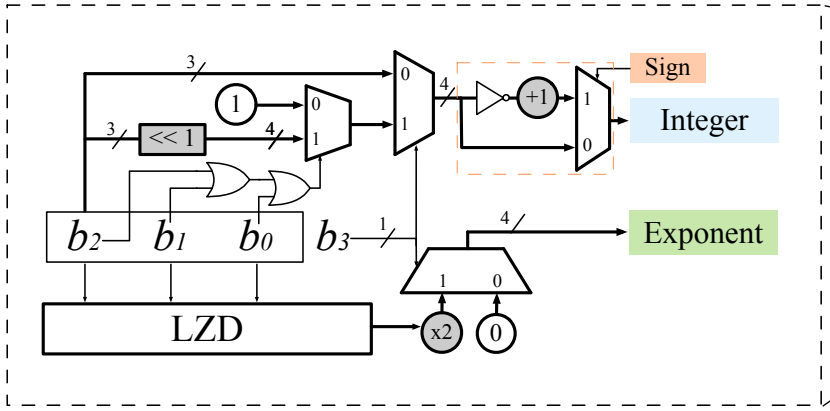
Challenge of ANT

- Tensor-wise Adaptive Numeric Data Type

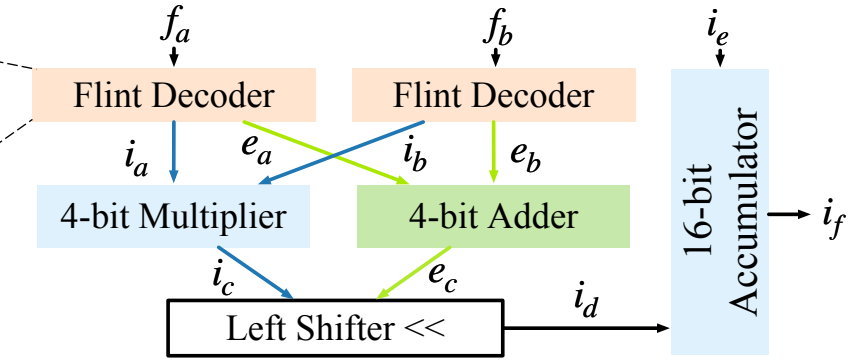


The input tensor and weight tensor may have different data types.
They cannot compute each other on existing hardware.

Inter-tensor ANT: Type Fusion



The 4-bit Flint decoder



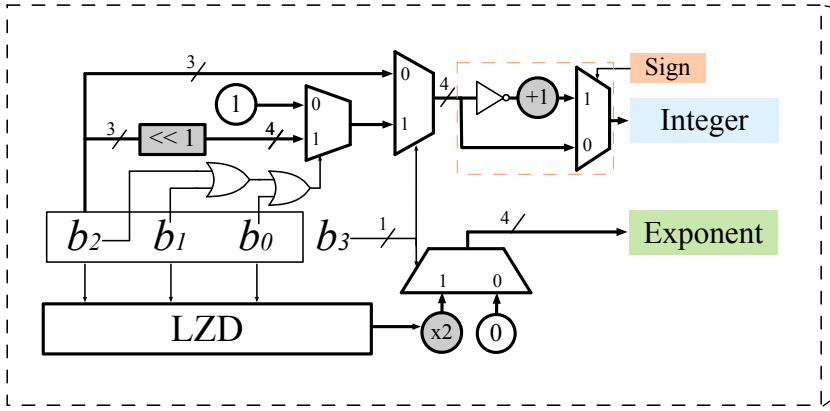
The 4-bit Type Fusion MAC unit

Embed the decoder into the Fixed-Point MAC unit.

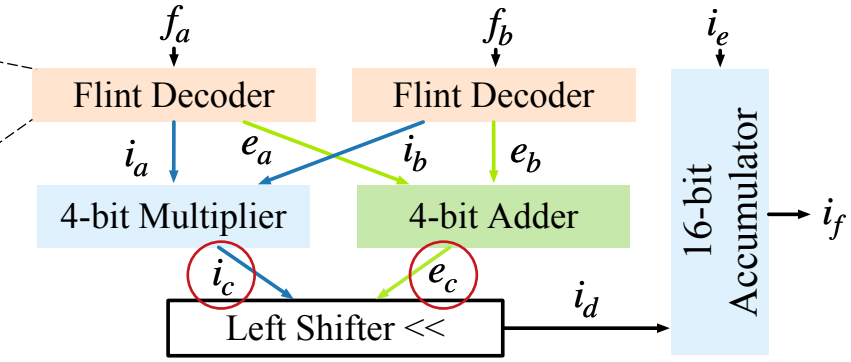
Type	Flint
Digits	x
Integer	
Exponent	
Real value	

Fixed-Point Type Fusion unit supports Flint

Inter-tensor ANT: Type Fusion



The 4-bit Flint decoder



The 4-bit Type Fusion MAC unit

Type	Flint
Digits	x
Integer	i
Exponent	e
Real value	$i << e$

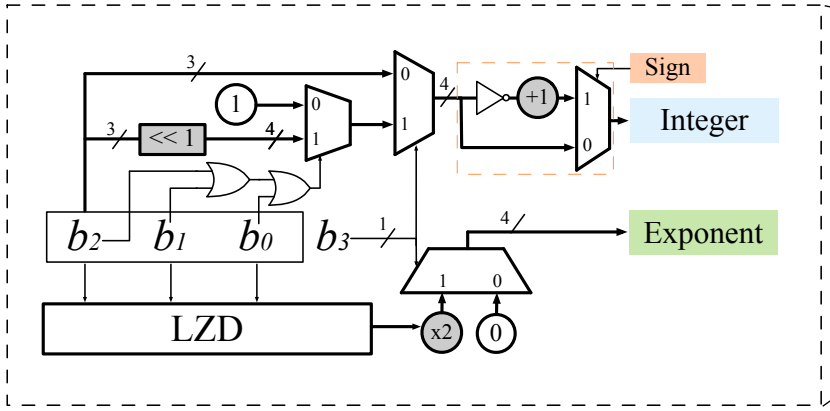
i is the integer.

e is the exponent.

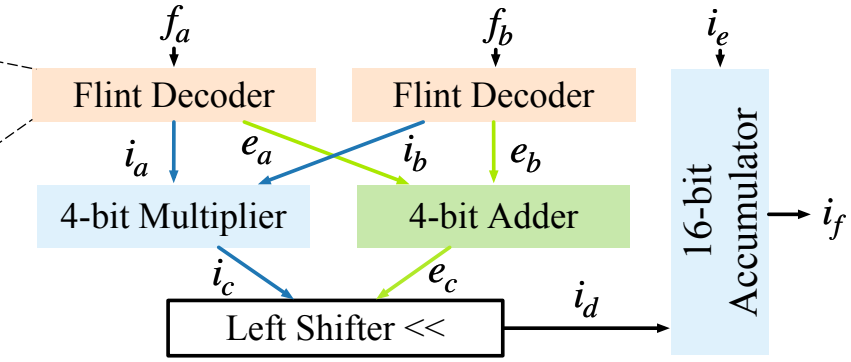
Use the fixed-point multiplier to multiply integers.
Use the fixed-point adder to add exponents.
Use the left shifter to get the final result.

Fixed-Point Type Fusion unit supports Flint

Inter-tensor ANT: Type Fusion



The 4-bit Flint decoder



The 4-bit Type Fusion MAC unit

Type	Flint	Int
Digits	x	x
Integer	i	x
Exponent	e	0
Real value	$i << e$	$x << 0$

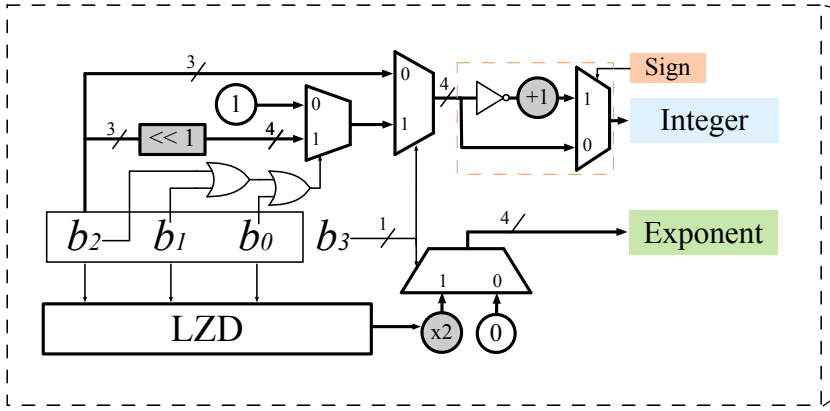
i is the integer.

e is the exponent.

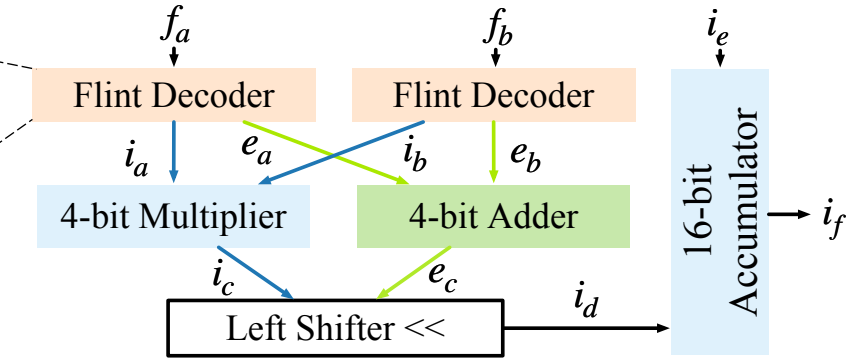
For Int, the exponent is always 0.

Fixed-Point Type Fusion unit supports Int, Flint

Inter-tensor ANT: Type Fusion



The 4-bit Flint decoder



The 4-bit Type Fusion MAC unit

Type	Flint	Int	PoT
Digits	x	x	x
Integer	i	x	1
Exponent	e	0	x
Real value	$i << e$	$x << 0$	$1 << x$

i is the integer.

e is the exponent.

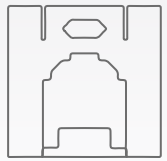
For Int, the exponent is always 0.

For PoT, the integer is always 1.

Fixed-Point Type Fusion unit supports Int, Flint, PoT

01

Background
&
Motivation



02

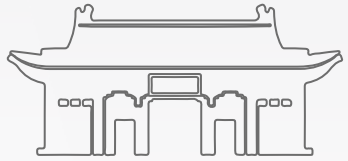
ANT Design

03

Integration

04

Evaluation

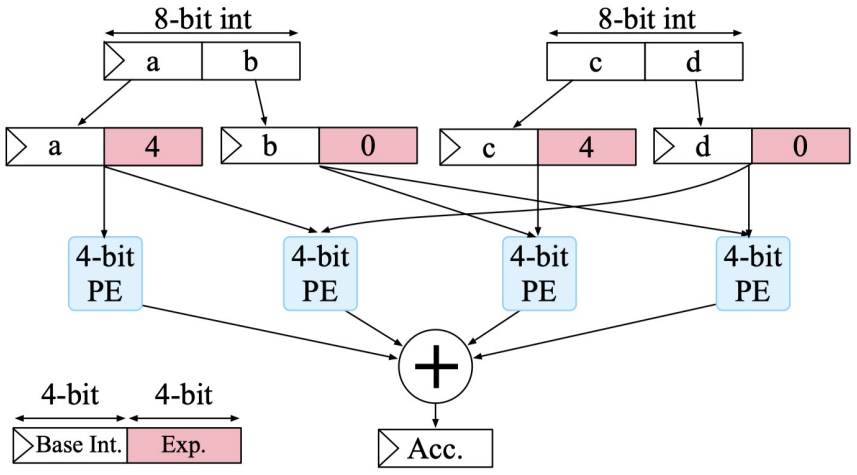


Integration Outline

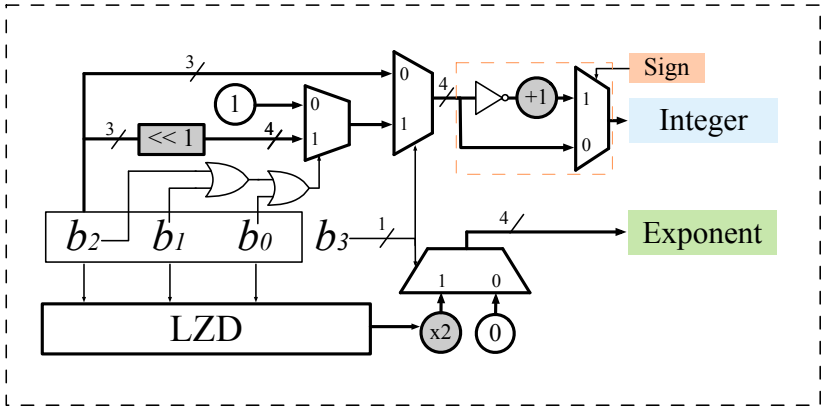
- Integration to MAC unit → Processing Element
- Integration to existing architectures → Architecture
- Integration to quantization framework → Quantization Framework

Processing Element Integration

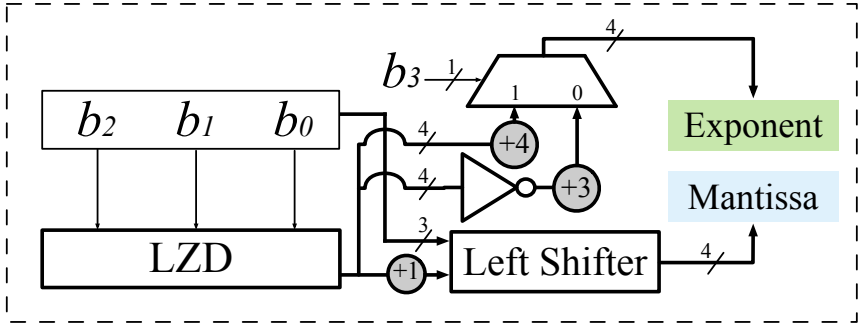
- Integration into MAC unit (Type Fusion)
  - Int-based PE (Fixed-Point)
  - Float-based PE (Float-Point)
  - Mixed Precision PE



The 8-bit Int MAC implementation via four 4-bit ANT MACs.



Fixed-Point decoder



Float-Point decoder

Also exploit the first-one decoding method.

Architecture Integration

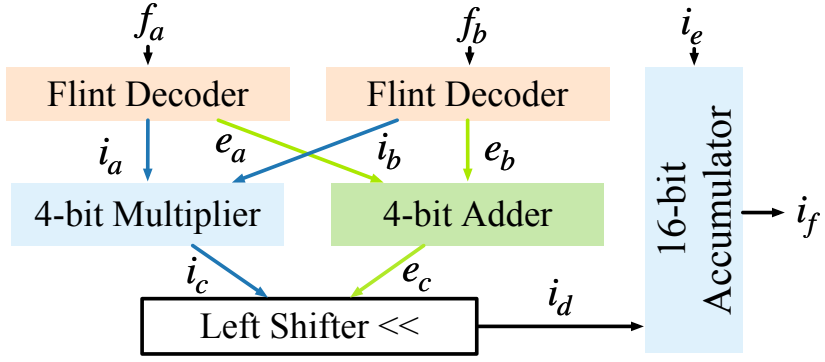
- Integration to existing architectures
  - GPU SIMD
  - Output Stationary Systolic Array
  - Weight Stationary Systolic Array
  - Instruction Set Extension

```
FMA.int16.int4.int4 {C, A, B};
```

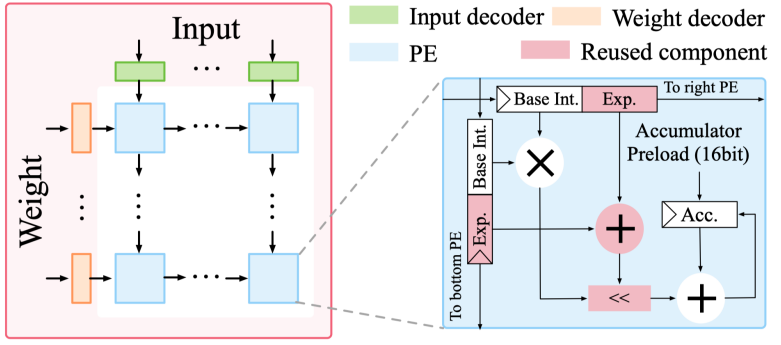


```
FMA.int16.flint4.int4 {C, A, B};
```

ANT can replace the original Int-based version with ANT type to generate the corresponding codes.



ANT can be directly embedded in GPU.



For systolic array, we only place the decoders along the boundary of PE array to mitigate the area overhead.

Quantization Framework Integration

Fine-tuning

Algorithm 1: Element-wise `flint` encoding algorithm.

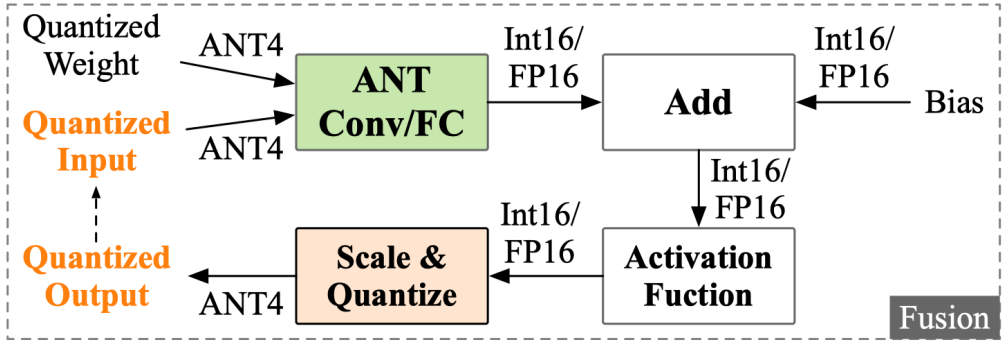
Input: Element, e; Bit-width, b; Scale factor, s.
Output: Quantized Element, q.

```
def FlintQuant(e, b, s):
    en = 2 × b; // Exponent number.
    e = IntQuantization(e, s, 0, 2^(en−2));
    if e == 0 then
        return 0;
    else
        i = ⌊log2(e)⌋ + 1;
        exp = GetExponent(b, i); // First-one exponent.
        mb = b − len(exp); // Mantissa bit.
        m = Round[(e/2^(i−1) − 1) × 2^mb];
        m = Binary(m);
        q = Concat(exp, m);
        return q;
```

Flint encoding algorithm. The flint encoding algorithm is an element-wise function that can be implemented efficiently in both hardware and software.

The algorithm can also be used for fine-tuning the quantized models in the training-aware quantization framework.

Inference



ANT-based quantized inference

For the Conv/FC layers, we use the low-bit quantized weights and input activations but keep the output activations the high precision.

The functions can fused to improve the efficiency.

01

Background
&
Motivation

02

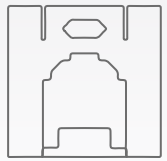
ANT Design

03

Integration

04

Evaluation



Evaluation Baseline and Tool

Quantization methods

Methods	Aligned	Mixed-precision
Int	Yes	No
BitFusion	Yes	Layer-wise
OLAccel	No	Element-wise
BiScaled	Yes	No
AdaptiveFloat	Yes	No
ANT	Yes	Layer-wise

Accelerator Implementation Tools

Tools	
Synopsys Design Compiler	Logic synthesis
CACTI	Memory simulation
DnnWeaver	End-to-End DNN simulator
DeepScaleTool	Technology Scaling

Verilog RTL, 28nm technology

DNN models

DNN models and datasets

Models	Datasets	Accuracy Loss
ResNet18	ImageNet	< 0.1%
ResNet50	ImageNet	< 0.1%
InceptionV3	ImageNet	< 0.1%
VGG16	ImageNet	< 0.1%
ViT	ImageNet	< 1%
BERT	GLUE (MNLI, CoLA, SST-2)	< 1%

Fine-tuning with PyTorch official checkpoints.

Iso-accuracy comparison

We adjust the mixed-precision ratio to make all models close to their original accuracy.

For CNN models: < 0.1% loss.
For Transformer models: < 1% loss.

Area Comparison

Architecture	Core			Buffer
	Component	Number	Area (mm^2)	
ANT	Decoder ($4.9\mu m^2$)	128	0.327	512 KB 4.2 mm^2
	4-bit PE ($79.57\mu m^2$)	4096		
BitFusion	4-bit PE	4096	0.326	
OLAccel8	4-bit & 8-bit PE	1152	0.320	
BiScaled	6-bit BPE	2560	0.328	
AdaFloat	8-bit PE	896	0.327	

Architecture	Decoder Ratio*
Int	0
AdaFloat	14.5%
BitFusion	~ 0
BiScaled	7.1%
OLAceel	71%
ANT (Ours)	0.2%

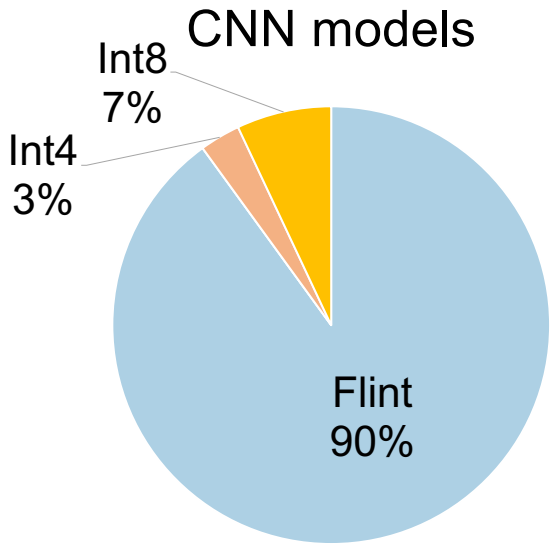
1. Int and BitFusion have no decoder.
2. OLAceel has significant overhead.
3. ANT overhead is only 0.2% for the systolic array.

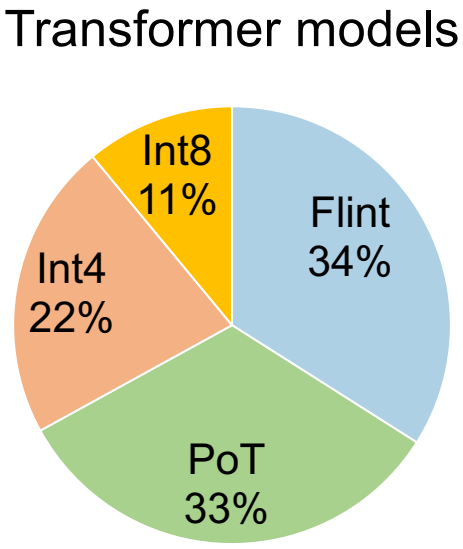
Decoder Ratio is the ratio of the decoder to the total area of the PE array.

Iso-area comparison

We implement all baselines with the same area. All accelerators have the same on-chip buffer configuration.

Proportion of Data Type



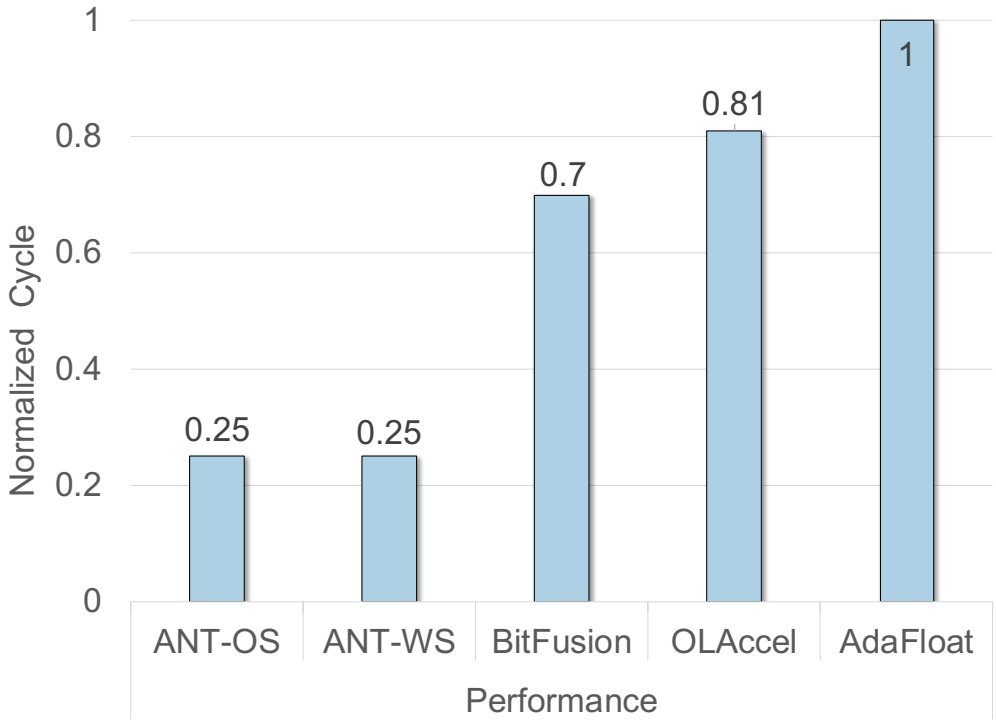


1. Flint is important for both CNN and Transformer models.

2. Most tensors of the CNN model are quantized by Flint.

3. Transformer models have relatively more complex tensor distributions.

Performance

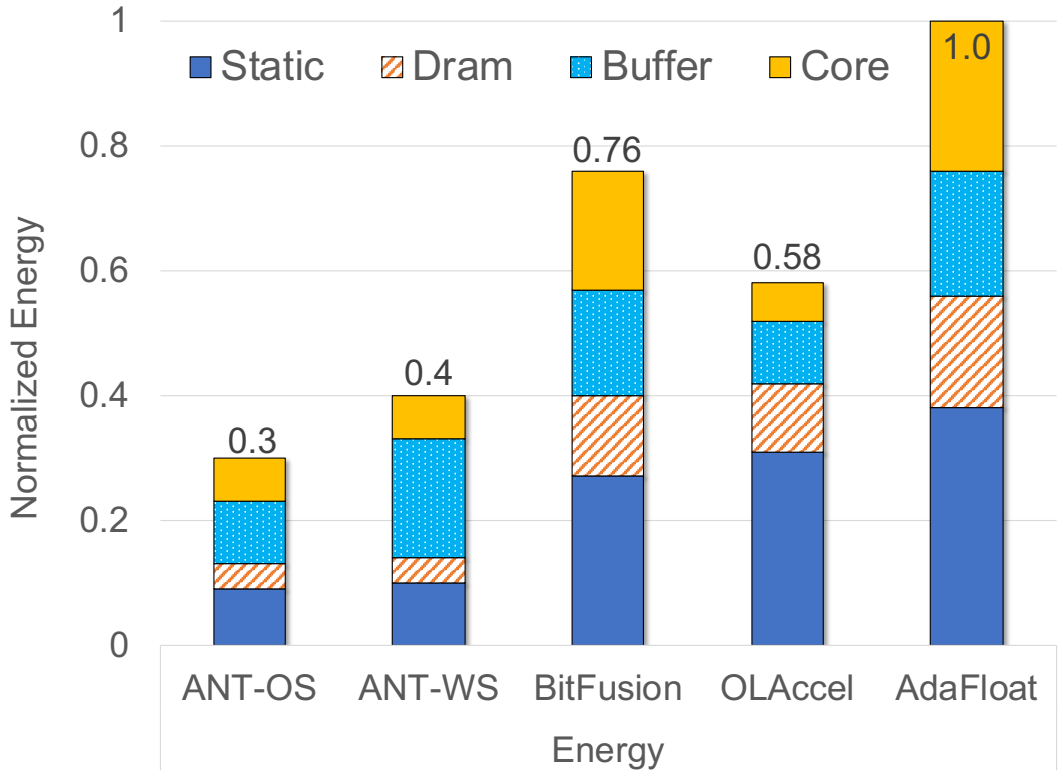


Under iso-area and iso-accuracy comparison:

1. ANT-OS and ANT-WS have very similar performances. They have the same type ratio.
2. OLAccel needs the additional outlier controller with significant overhead.

ANT achieves 2.8x performance improvement than BitFusion.

Energy



Under iso-area and iso-accuracy comparison:

Energy:

1. ANT-WS needs more buffer accesses for the high-precision output activation. ANT-WS spends more energy on accessing on-chip buffers.

2. OLAccel consumes less energy than BitFusion because it has more 4-bit values, which reduces the energy of DRAM and on-chip buffer.

ANT achieves 2.5x energy reduction than BitFusion.

Conclusion

- Multiple fixed-length data types.
- Intra-tensor adaptivity → Flint.
- Inter-tensor adaptivity → Type Fusion.

- The simplicity of the hardware implementation.
- Performance improvement and energy reduction.
- ANT design demonstrates 2.8× performance improvement and 2.5× energy reduction over the SOTA quantization accelerators.

上海交通大學

SHANGHAI JIAO TONG UNIVERSITY

Thanks!

饮 水 思 源　爱 国 荣 校