LLM Sys

11868/11968 LLM Systems
LLM Quantization
-- Basic methods

Lei Li

Carnegie Mellon University
Language Technologies Institute

Ack: Kath Choi, Aashiq Muhamed, Zhen Wu, Hongyi Jin, Wayne Wang

Recap of Model Parallel Training

- Pipeline Parallelism
  - split by layers (horizonal split)
  - eliminate the bubbles (idle)
  - interleaving forward/backward
- Tensor Parallelism
  - Split the matrix computation

Today’s Topic

- Low precision numbers in computer
- Basic Quantization Methods

LLM Training and Inference are Costly!

Llama-70B

39.3M H100-80GB GPU hours to train

requires 140GB GPU memory for inference

Deepseek V3 (671GB)

2.8M H800 GPU hours to train

requires > 400GB GPU memory for inference

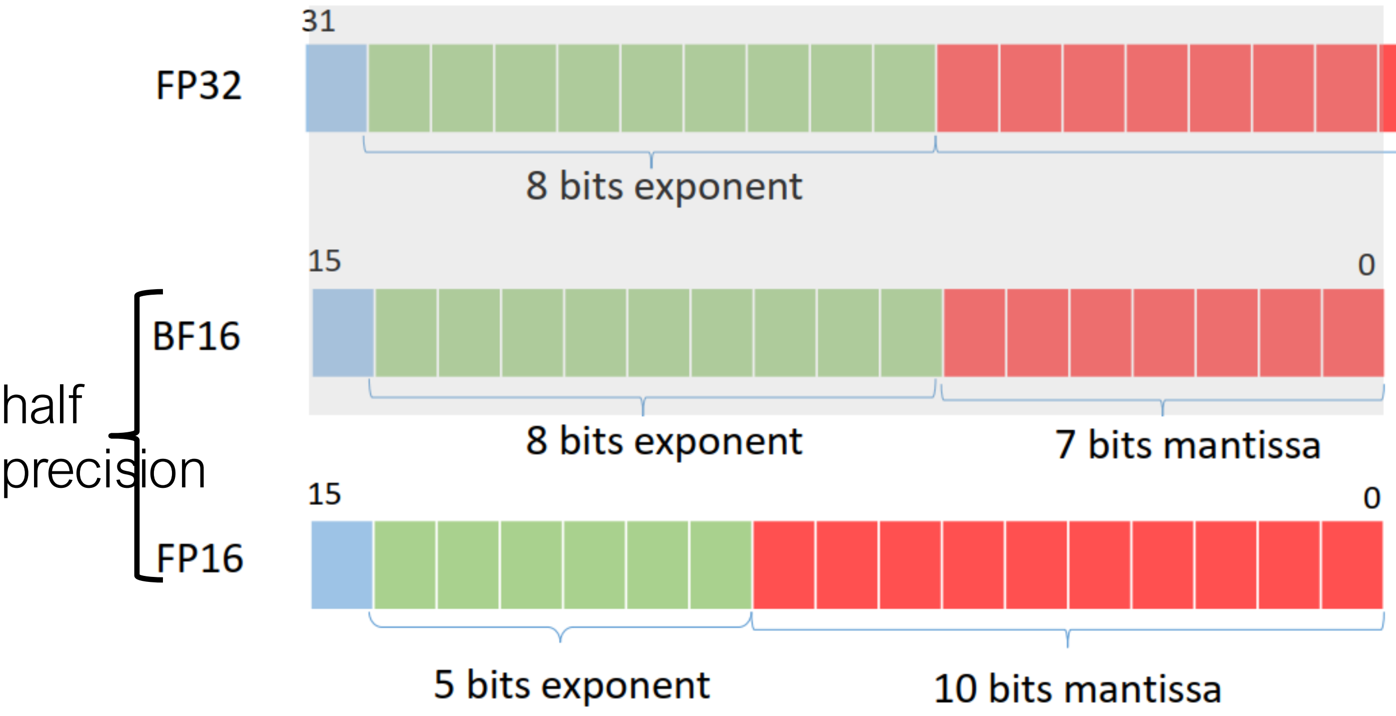
Model Quantization

- Use low-bit precision to store parameters and layer outputs
- Quantization can
  - reduce memory ➔ larger batch size
  - speed up calculation, more operations in one cycle
- Cons: potentially reduce accuracy

A White Paper on Neural Network Quantization (Nagel et al., 2021)

Precision Formats

$$\pm 1.(mantissa) * 2^{\{a-2^{\{exponent-1\}}\}}$$



Data Type	Dynamic Range	Precision
FP32	$-3.4\times10^{38} \sim 3.4\times10^{38}$	7×10^{-46}
BF16	$-3.4\times10^{38} \sim 3.4\times10^{38}$	3.05×10^{-5}
FP16	$-8.59\times10^{9} \sim 8.59\times10^{9}$	2.98×10^{-8}

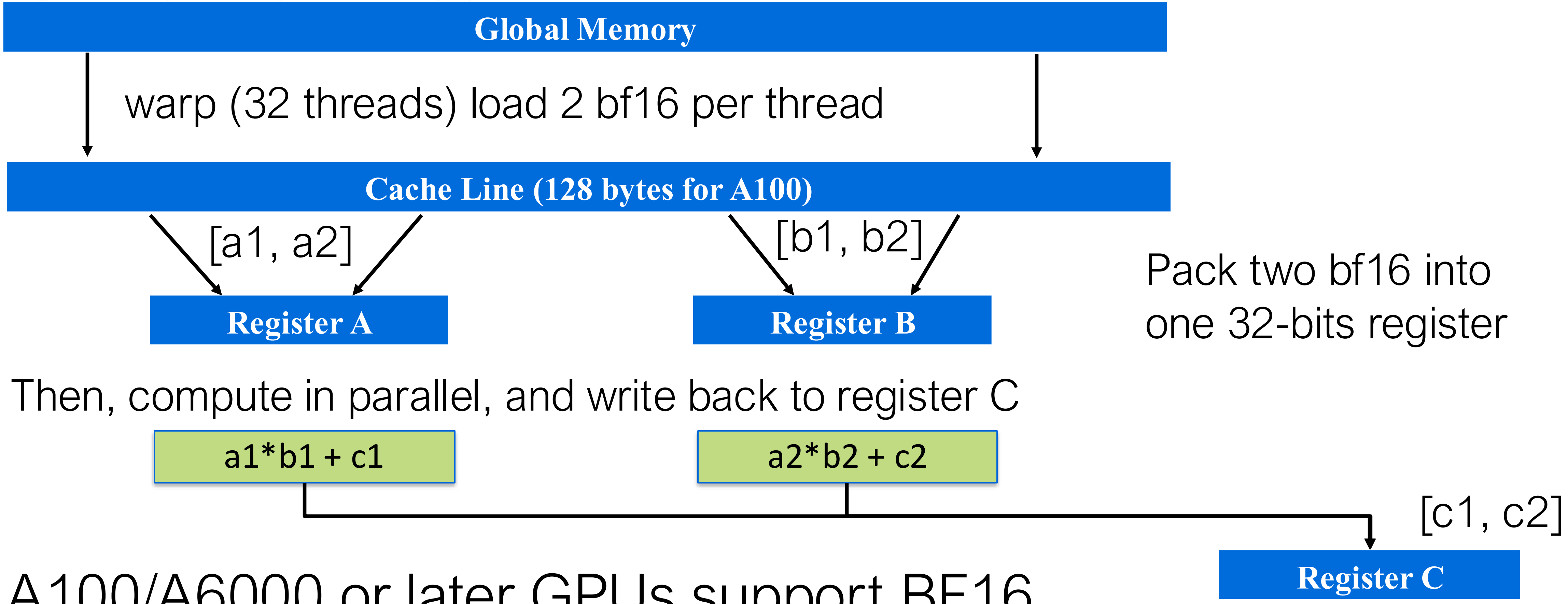
BF16 favors Dynamic Range than Precision.

INT8 range: -128~127, Precision: integer

Image from https://deeprec.readthedocs.io/en/latest/BFloat16.html

BF16/FP16 Calculations are faster!

HFMA2: Half-precision Fused Multiply-Add for 2 elements in one cycle (2x speedup)



A100/A6000 or later GPUs support BF16

CUDA APIs for Half Precision

Performs half2 vector addition in round-to-nearest-even mode.

```
__device__ __half2 __hadd2(const __half2 a, const __half2 b)
```

Performs half2 vector fused multiply-add in round-to-nearest-even mode.

```
__device__ __half2 __hfma2(const __half2 a, const __half2 b, const
__half2 c)
```

Direct Quantization Approach

- Using lower precision
  - converting parameters from FP32 to INT8 or INT4
  - perform all computation in lower prevision.

- Reduce model accuracy:
  - Loss of Precision ➔ accumulate quantization noise
  - Range mismatch ➔ values are clipped and lead to information loss
  - Quantization error ➔ rounding errors

Quantize a number

- Absmax quant

$$\mathbf{X}_{\text{quant}} = \text{round}\left(\frac{127}{\max|\mathbf{X}|} \cdot \mathbf{X}\right)$$

$$\mathbf{X}_{\text{dequant}} = \frac{\max|\mathbf{X}|}{127} \cdot \mathbf{X}_{\text{quant}}$$

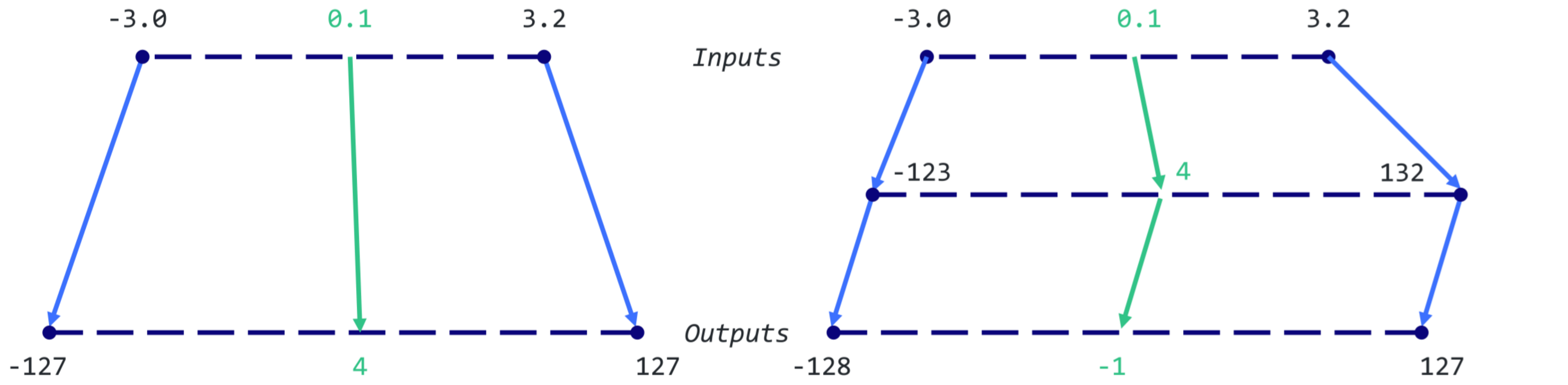
- Zero-point quant

$$\text{scale} = \frac{255}{\max(\mathbf{X}) - \min(\mathbf{X})}$$

$$\text{zeropoint} = -\text{round}(\text{scale} \cdot \min(\mathbf{X})) - 128$$

$$\mathbf{X}_{\text{quant}} = \text{round}\left(\text{scale} \cdot \mathbf{X} + \text{zeropoint}\right)$$

$$\mathbf{X}_{\text{dequant}} = \frac{\mathbf{X}_{\text{quant}} - \text{zeropoint}}{\text{scale}}$$



Code implementation for quant(.)

```
import torch
def absmax_quantize(X):
	# Calculate scale
	scale = 127 / torch.max(torch.abs(X))
	# Quantize
	X_quant = (scale * X).round()
	# Dequantize
	X_dequant = X_quant / scale
	return X_quant.to(torch.int8), X_dequant
```

```
def zeropoint_quantize(X):
	# Calculate value range (denominator)
	x_range = torch.max(X) - torch.min(X)
	x_range = 1 if x_range == 0 else x_range
	# Calculate scale
	scale = 255 / x_range
	# Shift by zero-point
	zeropoint = (-scale * torch.min(X) - 128).round()
	# Scale and round the inputs
	X_quant = torch.clip((X * scale + zeropoint).round(), -128,
	127)
	# Dequantize
	X_dequant = (X_quant - zeropoint) / scale
	return X_quant.to(torch.int8), X_dequant
```

Direct Quantization Colab

https://colab.research.google.com/drive/1DPr4mUQ92Cc-xf4GgAaB6dFcFnWIvqYi?usp=sharing

Today's Topic

- Low precision numbers in computer
- Basic Quantization Methods

Model Quantization Approaches

Quantization
- during training → expensive re-training / finetuning
- post training

Model Quantization Approaches

- Quantization
  - ~~during training~~
  - post training
    - preserve accuracy
      - AdaQuant (Hubara et al., 2021)
      - BRECQ (Li et al., 2021)
      - OBQ (Frantar et al., 2022)
    - scale to large parameters
      - ZeroQuant (Yao et al., 2022)
      - LLM.int8() (Dettmers et al., 2022)

Model Quantization Approaches

hard to scale to large parameters (billions)

- Quantization
  - ~~during training~~
  - post training
    - preserve accuracy (by quantizing each individual / consecutive layers)
      - AdaQuant (Hubara et al., 2021)
      - BRECQ (Li et al., 2021)
      - OBQ (Frantar et al., 2022)
    - scale to large parameters
      - ZeroQuant (Yao et al., 2022)
      - LLM.int8() (Dettmers et al., 2022)

Basic Approach: Layer-Wise Quantization

- Find the quantized matrix $\widehat{W}$ that minimizes the linear layer's output.

- Performs quantization layer-by-layer linear projection

$$\underset{\widehat{W}}{\operatorname{argmin}} \left\| WX - \widehat{W}X \right\|_2^2$$

W: linear projection weights (e.g. in FFN and attention)
X: layer input

- Limitation: could still lead to accumulation of quantization error

AdaQuant

- minimize the error between the quantized / full-precision layer outputs for each layer

- adding continuous V to W and quantize

$$(\widehat{\Delta_w}, \widehat{\Delta_x}, \hat{V}) = \underset{\Delta_w \, \Delta_x, V}{\operatorname{argmin}} \left\| WX - Q_{\Delta_w}(W + V) \cdot Q_{\Delta_x}(X) \right\|^2$$

quantized results

the quantization step size Δ_w and Δ_x can be set or learned.

Hubara et al. Improving Post Training Neural Quantization: Layer-wise Calibration and Integer Programming. 2020.

Is Quantization Accurate?

MobileNet-v2	Float Model	Direct Quantized Model	Quantized Model with ADAQUANT
Model Size	8.4 MB	2.3 MB	2.4 MB
P rec@1	65.4 %	1.7 %	52.3 %
P rec@5	85.7 %	5.6 %	75.7 %

Why is Quantizing LLMs Difficult?

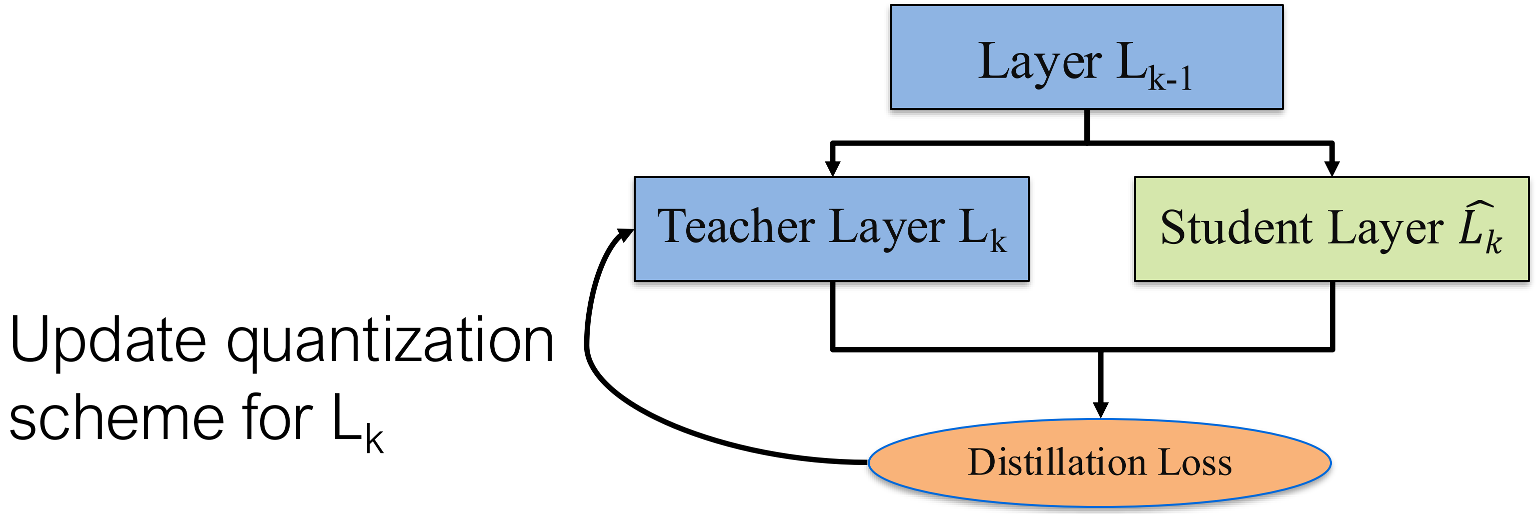
- Quantization
  - ~~during training~~
  - post training
    - preserve accuracy
      - AdaQuant (Hubara et al., 2021)
      - BRECQ (Li et al., 2021)
      - OBQ (Frantar et al., 2022)
    - scale to large parameters (by rounding weights to the nearest quantization level)
      - ZeroQuant (Yao et al., 2022)
      - LLM.int8() (Dettmers et al., 2022)

accuracy loss when lower-bit precision (ex. 3, 4 bits per parameter)

ZeroQuant

- Layer-by-layer knowledge distillation
  - Use the original model as Teacher
  - the quantized model is student

$$\mathcal{L}_{LKD,k} = \left| L_k \cdot L_{k-1} \cdot L_{k-2} \cdot \cdots \cdot L_1(X) - \hat{L}_k \cdot L_{k-1} \cdot L_{k-2} \cdot \cdots \cdot L_1(X) \right|^2$$



Yao et al. ZeroQuant: Efficient and Affordable Post-Training Quantization for Large-Scale Transformers. Neurips 2022.

ZeroQuant

- Quantization-Optimized Transformer Kernels (fusion)

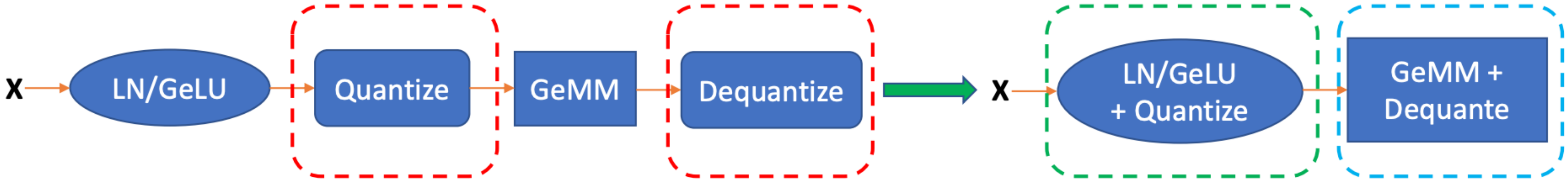


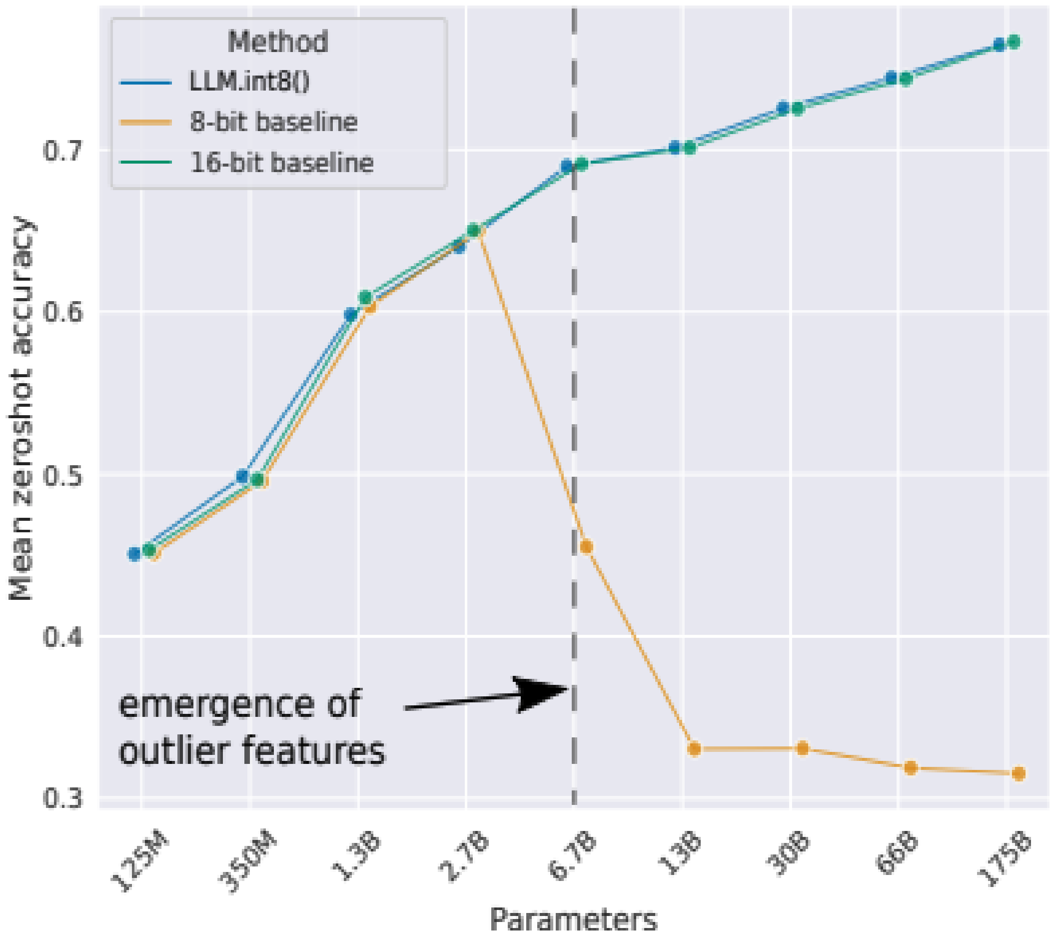
Figure 2: The illustration of normal (left) and our fused (right) INT8 GeMM.

- The scalability is verified up to 20B models (GPT-NeoX$_{20B}$)
- At 1.3B scale, computation time is ~3 hours
  - but slower than GPTQ (x100 larger in ~4 hours)
- integrated in Deepspeed

Yao et al. ZeroQuant: Efficient and Affordable Post-Training Quantization for Large-Scale Transformers. Neurips 2022.

LLM.int8()

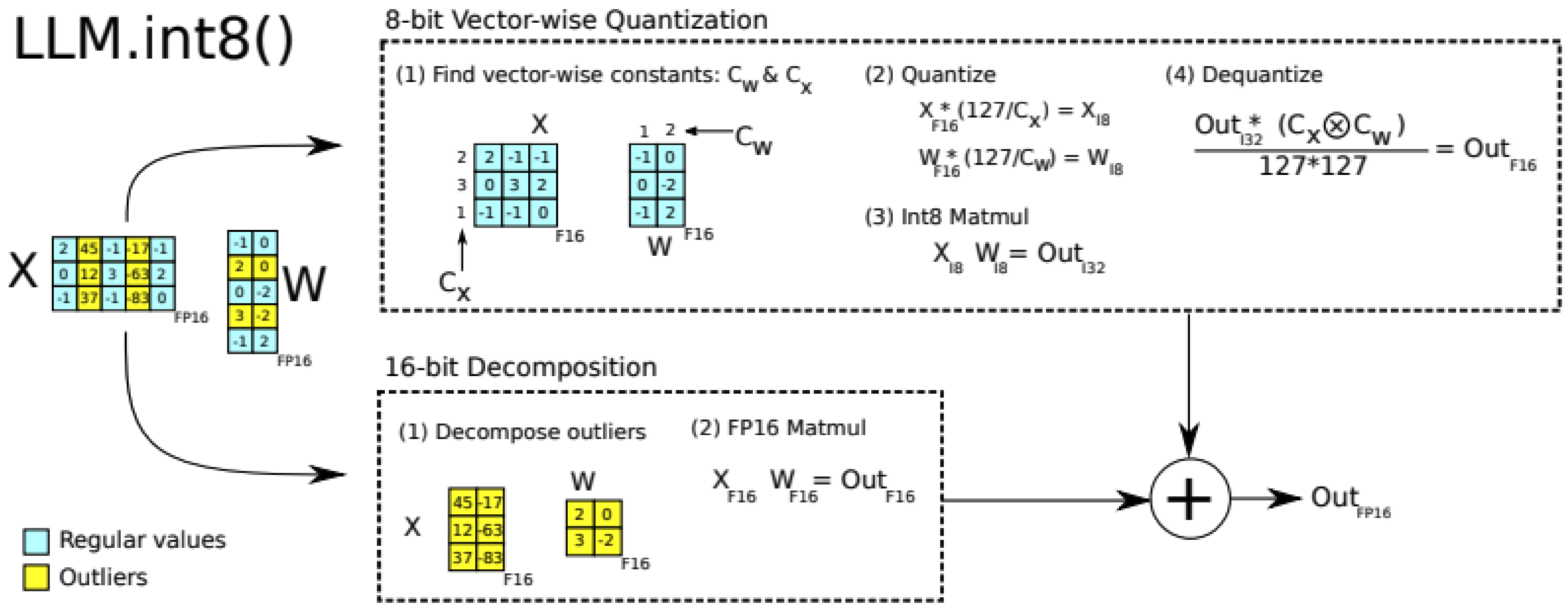
- Using 8-bit quantization for matrix multiplications
- But, extreme outliers in features (activation values)
  - need for wider numerical ranges
  - Quantize all parameters without distinguishing them separately can result in accuracy degradation



LLM.int8(): 8-bit Matrix Multiplication for Transformers at Scale. Dettmers et al. Neurips 2022.

LLM.int8()

- Keep outliers in higher precision (FP16) while quantizing the rest (8bit)
- Outliers: large magnitude (>= 6.0) , affects >= 25% layers, and affects >= 6% sequence dimensions



LLM.int8(): 8-bit Matrix Multiplication for Transformers at Scale, Dettmers et al. Neurips 2022.

Summary

- low-bit number representation in computer
  - BF16: 16-bit half precision floating point numbers, better for ML
  - int8
- Direct quantization
  - absmax: linearly scale according to max abs value
  - zero-point: finding zero-point and scale
- Layer-wise quantization approaches
  - AdaQuant
  - KD: ZeroQuant
  - LLM.int8()

Next

- Scaling Quantization for large models: GPTQ