Accelerating Transformers with FP8

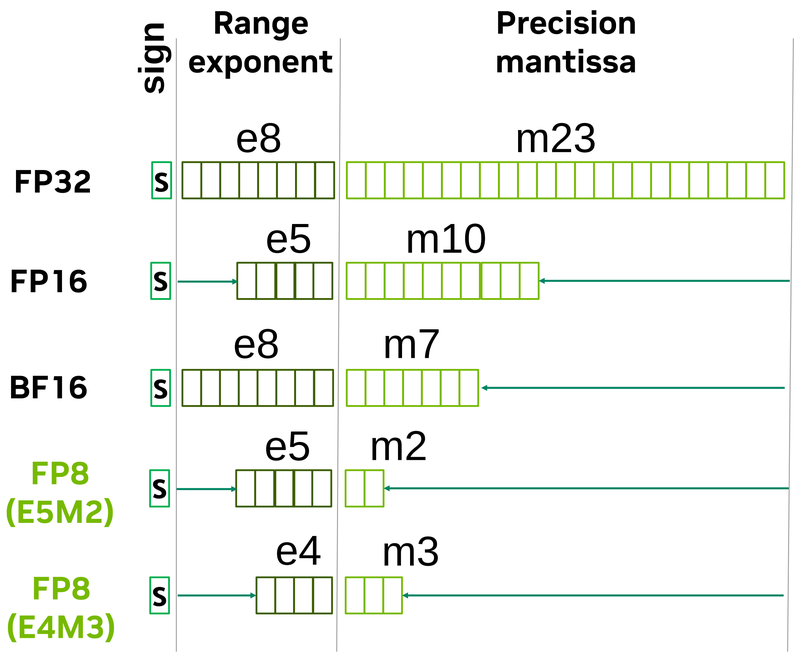
Griffin Lacey | 2023 May 04

Some History on Tensor Cores

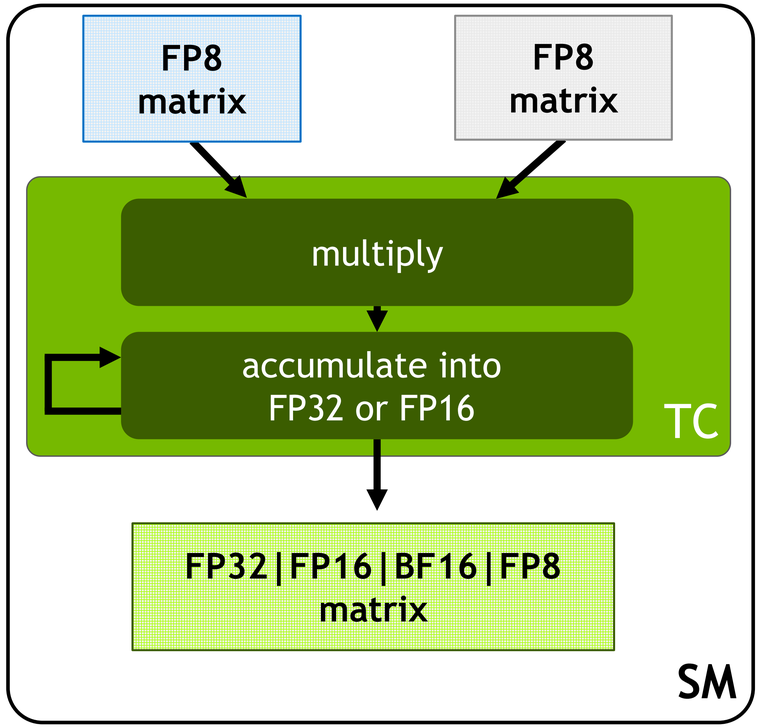
- Tensor Cores are specialized hardware execution units that perform matrix multiplications
  - 8x - 64x faster than scalar instructions in “CUDA Cores”
  - More energy efficient
- First introduced in Volta GPU microarchitecture (2017)
  - Higher densities of Tensor Cores in datacenter GPUs (Tesla)

	Ada	Hopper	Ampere	Turing	Volta
Supported Tensor Core precisions	FP64, TF32, bfloat16, FP16, FP8, INT8	FP64, TF32, bfloat16, FP16, FP8, INT8	FP64, TF32, bfloat16, FP16, INT8, INT4, INT1	FP16, INT8, INT4, INT1	FP16
Supported CUDA® Core precisions	FP64, FP32, FP16, bfloat16, INT8	FP64, FP32, FP16, bfloat16, INT8	FP64, FP32, FP16, bfloat16, INT8	FP64, FP32, FP16, INT8	FP64, FP32, FP16, INT8

Inside FP8 Tensor Cores



Allocate 1 bit to either range or precision



Support for multiple accumulator and output types

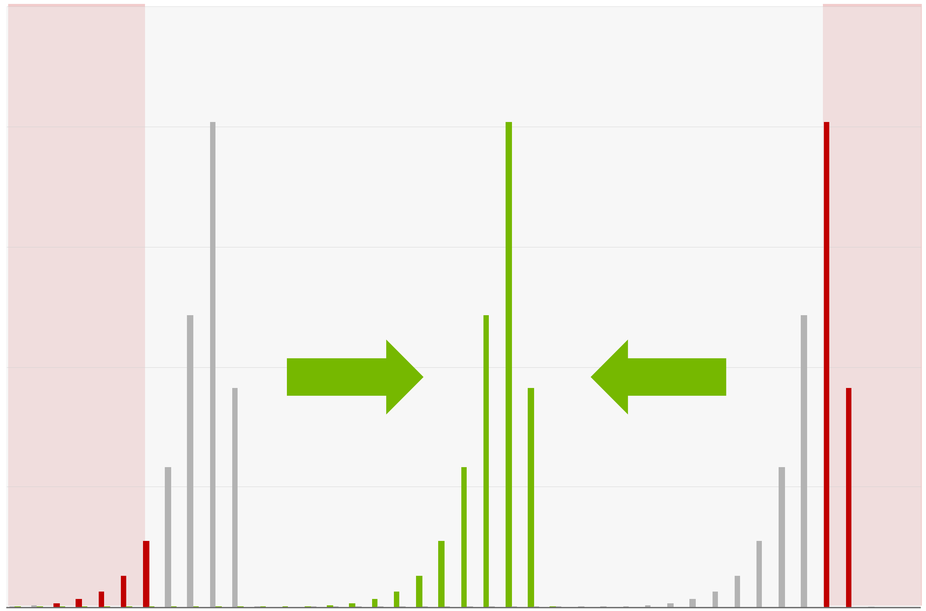
Why use FP8?

- Accelerate math-intensive operations
  - On H100 using FP8 with Tensor Cores has 60x more theoretical throughput than FP32 with CUDA cores
- Accelerates memory-intensive operations and reduces storage requirements
  - FP8 requires 25% of the memory of FP32
- Facilitates preparation for inference
  - Many models are deployed in low precision

Mixed Precision Recipe

The FP16 Way

- Partition the DL graph into safe and unsafe ops
  - Safe ops benefit from reduced precision and have dynamic ranges are similar from input to output
- During forward pass compute multiplications in FP16 and accumulations in FP32 for safe ops
- During backward pass use loss scaling factor to avoid gradient under/overflows



Mixed Precision Recipe

The FP8 Way

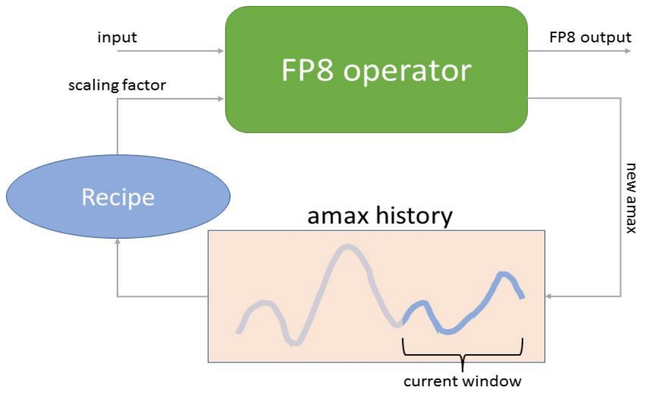
- Partition the DL graph into safe and unsafe ops
  - Unsafe ops don't necessarily need to be FP32, can be FP16/BF16

- During forward pass use E4M3 and in backward pass use E5M2
  - E4M3 dynamic range: 18 powers of 2
  - E5M2 dynamic range: 32 powers of 2

Dynamic range (in powers-of-two) needed for different tensors in GEMMs.

Network	Activations	Weights	Activation Gradients	Weight Gradients
GPT 126M	10	10	17	11
Transformer-XL	8	8	11	8
ResNet50	7	5	16	9
ResNet18	5	7	14	9

- During backward pass use per-tensor scaling factors
  - Pick scaling factor based on window of past N instances of the tensor



What is Considered Safe?

- Operations that are safe in FP8
  - Convolutions
  - Linear layers in Transformer blocks
  - BatchNorm has been tested for specific cases (e.g. ResNet50)

- Operations that should remain in higher precision
  - Most nonlinear functions and normalizations (GeLU, Softmax, …)
  - Residual connections in LLMs
  - Loss functions

- Context dependent – some networks or tasks may tolerate more layers in FP8

```
import torch
import transformer_engine.pytorch as te
from transformer_engine.common import recipe

# Set dimensions.
in_features = 768
out_features = 3072
hidden_size = 2048

# Initialize model and inputs.
model = te.Linear(in_features,
                  out_features,
                  bias=True)
inp = torch.randn(hidden_size,
                  in_features,
                  device="cuda")

# Create FP8 recipe.
fp8_recipe = recipe.DelayedScaling()

# Enable autocasting to FP8.
with te.fp8_autocast(enabled=True,
                     fp8_recipe=fp8_recipe):
    out = model(inp)

loss = out.sum()
loss.backward()
```

Transformer Engine

- An open-source library implementing the FP8 recipe for Transformer building blocks
- Optimized for FP8 and other datatypes
- Supports PyTorch, with JAX and TF support coming soon
- Composable with the native framework operators
- Supports model and sequence parallelism
- **https://github.com/NVIDIA/TransformerEngine**

Image Classification

Convolutional Networks

Model	16-bits	FP8
ResNet18	70.58	70.12
ResNet34	73.84	73.72
ResNet50	76.65	76.61
ResNet101	77.51	77.48[1]
ResNext50	77.68	77.62
Wide ResNet50	78.13	77.90[1]
InceptionV3	77.23	77.07[1]
Xception	79.46	79.24[1]
DenseNet121	75.59	75.39[1]
DenseNet169	76.97	76.94[1]
Dilated ResNet C-26	75.22	75.04[1]
Dilated ResNet C-42	76.80	76.52[1]
Dilated ResNet A-50	78.16	78.12[1]
MobileNet V2	71.65	71.04[1]

[1]Experiments on different FP8 recipe choices

Vision Transformers

Model	16-bits	FP8
DeiT Tiny	72.69	72.52
DeiT Small	80.08	79.75[1]
ViT Tiny	71.61	71.24
ViT Small	77.19	77.24
ViT Base	80.27	80.20
ViT Large	78.38	78.39
Swin-V1 Tiny	81.13	81.16
Swin-V1 Small	83.02	83.07
Swin-V1 Base	83.50	83.42
Swin-V2 Tiny	82.79	82.76
Swin-V2 Small	84.30	84.12
Swin-V2 Base	84.50	84.60
GC-ViT Tiny	82.32	82.88
GC-ViT Small	84.03	83.98
GC-ViT Base	84.42	84.46
FAN Tiny	78.65	78.75
FAN Small	82.57	82.56
FAN Base	83.41	83.14

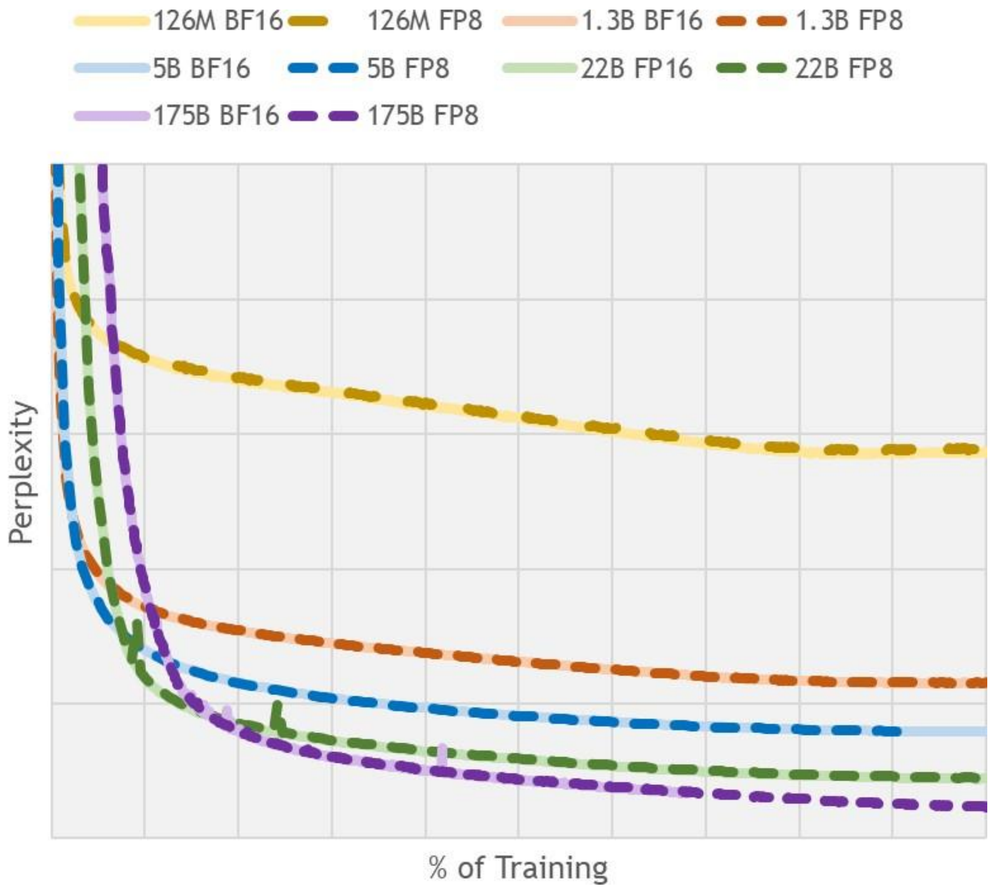
FP8 Formats for Deep Learning, Micikevicius et. al.

Natural Language Processing

Network	Metric	16-bits	FP8
GNMT	BLEU	24.83	24.65[1]
Vaswani Base	BLEU	26.87	26.83[1]
Vaswani Large	BLEU	28.43	28.35[1]
Transformer-XL Base	PPL*	22.71	22.76
Transformer-XL Large	PPL*	17.90	17.85
Megatron BERT Base	Loss*	1.352	1.357[1]
Megatron BERT Large	Loss*	1.163	1.167[1]
JoC BERT Large	F1	89.89	90.23
T5 Base	F1	91.68	91.88
T5 Large	F1	93.41	93.66
T5 Base	Rouge	42.88	42.88
T5 Large	Rouge	43.84	43.64
GPT 126M	PPL*	19.36	19.50
GPT 357M	PPL*	14.07	14.17
GPT 1.3B	PPL*	10.77	10.78
GPT 5B	PPL*	8.95	8.98
GPT 22B	PPL*	7.21	7.24
GPT 175B	PPL*	6.65	6.68

[1]Experiments on different FP8 recipe choices

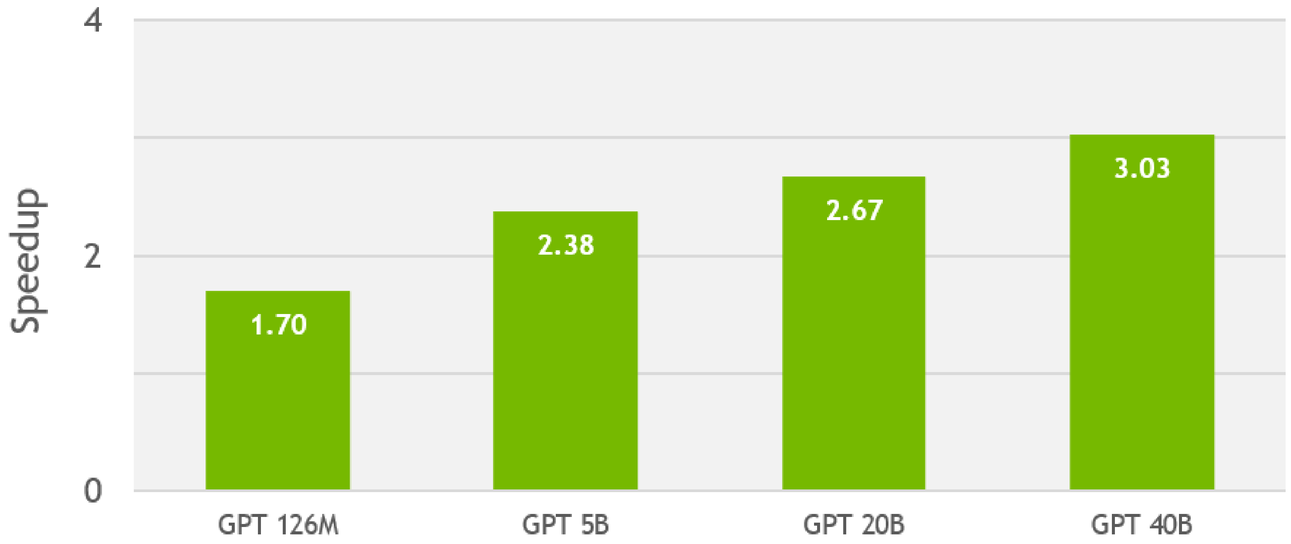
*Lower means better



Results based on running on A100 (16-bit Tensor Cores) with FP8 IO and on H100 FP8 Tensor Cores

DL TRAINING PERFORMANCE

- LLMs
  - Up to 3x faster with H100 FP8 than A100 BF16/FP16



Conclusions

- What's needed for accuracy?
  - Transformer Engine Recipes
    - Per-tensor scaling
    - Both FP8 types (E5M2 and E4M3)
    - Mixed precision
- What's needed for performance?
  - H100 GPU
    - FP8 Tensor Cores for faster math
    - FP8 I/O to reduce bandwidth pressure
- You should consider using if you:
  - Are training large DL models (e.g. LLMs) and want further speedup on H100
  - Already use 16-bit mixed-precision on Ampere and other GPUs


NVIDIA