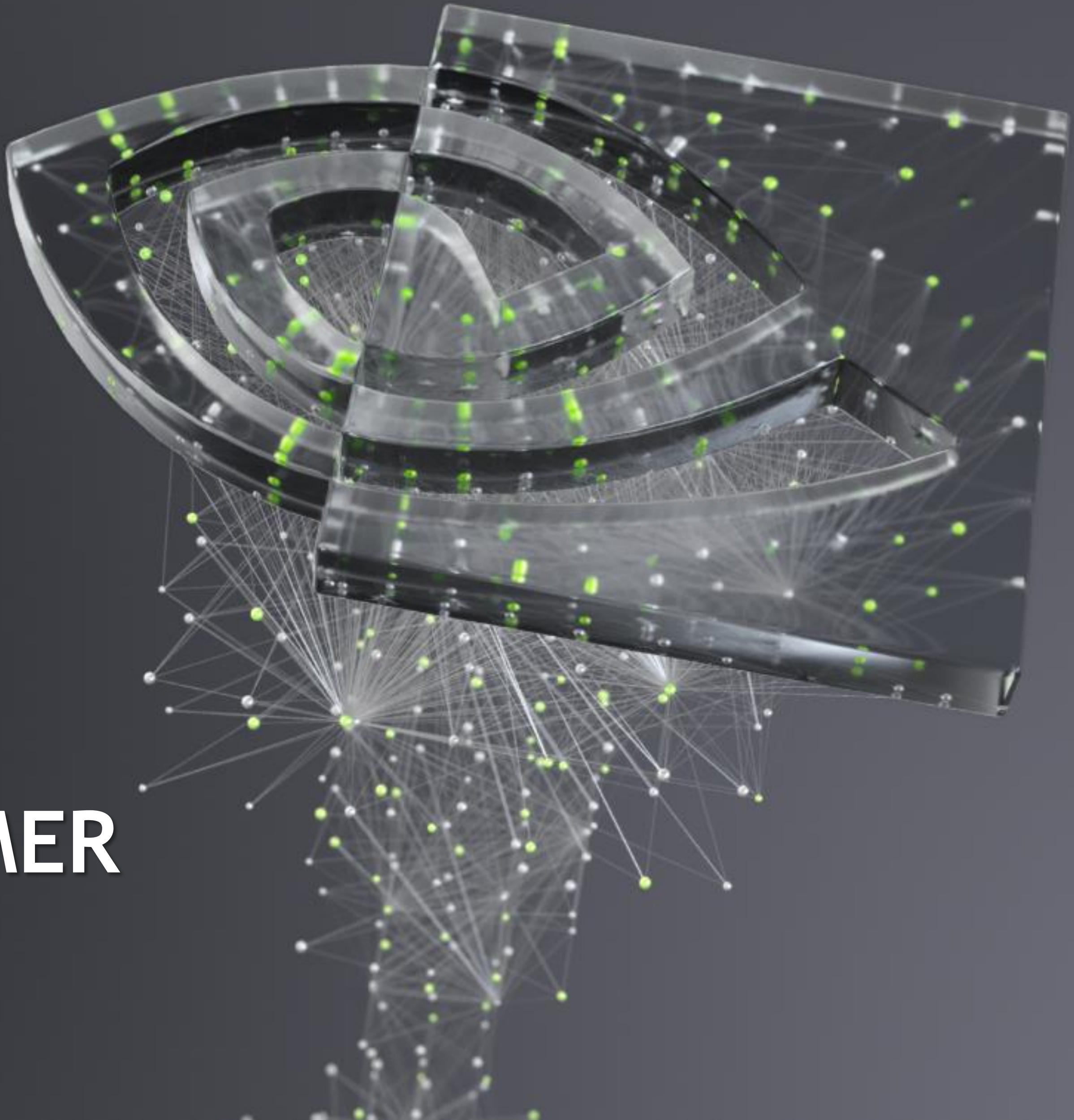
FASTER TRANSFORMER

Bo Yang Hsueh, 2020/03/12



AGENDA

WHAT IS FASTER TRANSFORMER

WHAT IS FASTER TRANSFORMER

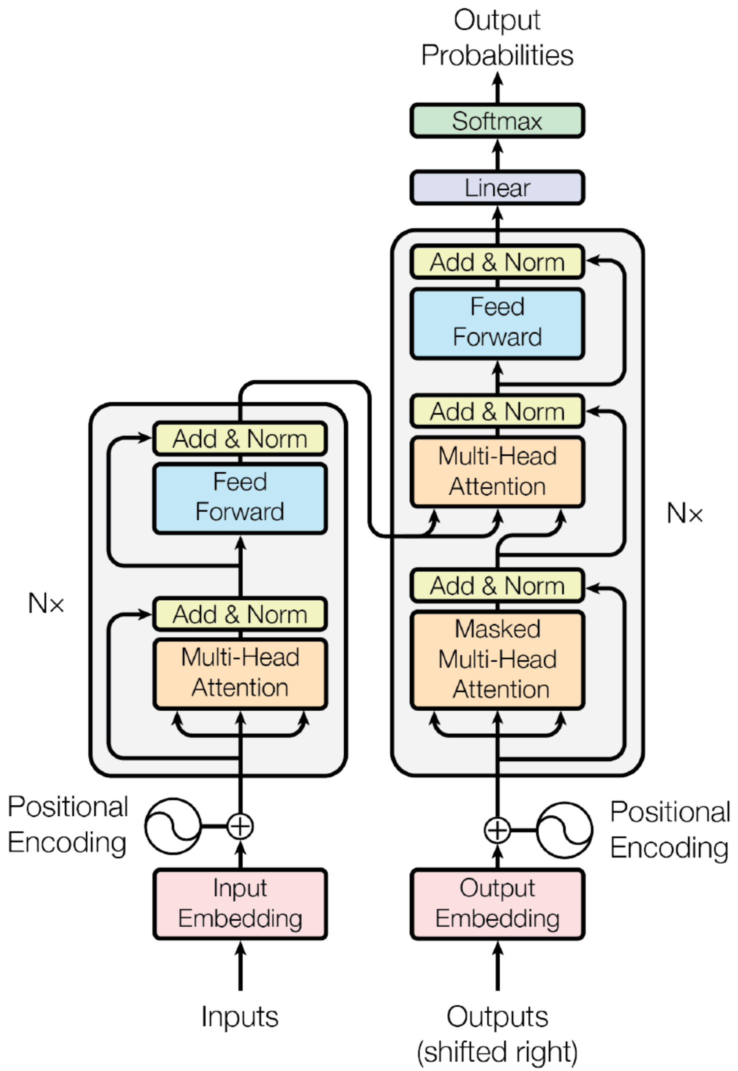
What is Transformer

Proposed by "Attention Is All You Need" [1]

The major component in BERT [2]

Become a popular model architecture in NLP

Problem: Model too large
Hard to satisfy the latency requirement in real applications



Transformer architecture [1]

[1] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł. and Polosukhin, I., 2017. Attention is all you need. In Advances in neural information processing systems (pp. 5998-6008).
[2] Devlin, J., Chang, M.W., Lee, K. and Toutanova, K., 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805.

WHAT IS FASTER TRANSFORMER

Features of Encoder and Decoder

- For Encoder-Decoder architecture, Decoder consumes most time (about 90%).
  - Decoder is the bottleneck
  - Optimizing the Decoder is more important (by Amdahl's law)
- However, pure Encoder architecture has good performance in many applications.
  - So, optimizing the Encoder is also necessary.

	workload	How many times	Friend to GPU
Encoder	Compute many words	One (or few)	More
Decoder	Compute one word	sequence length times	Less

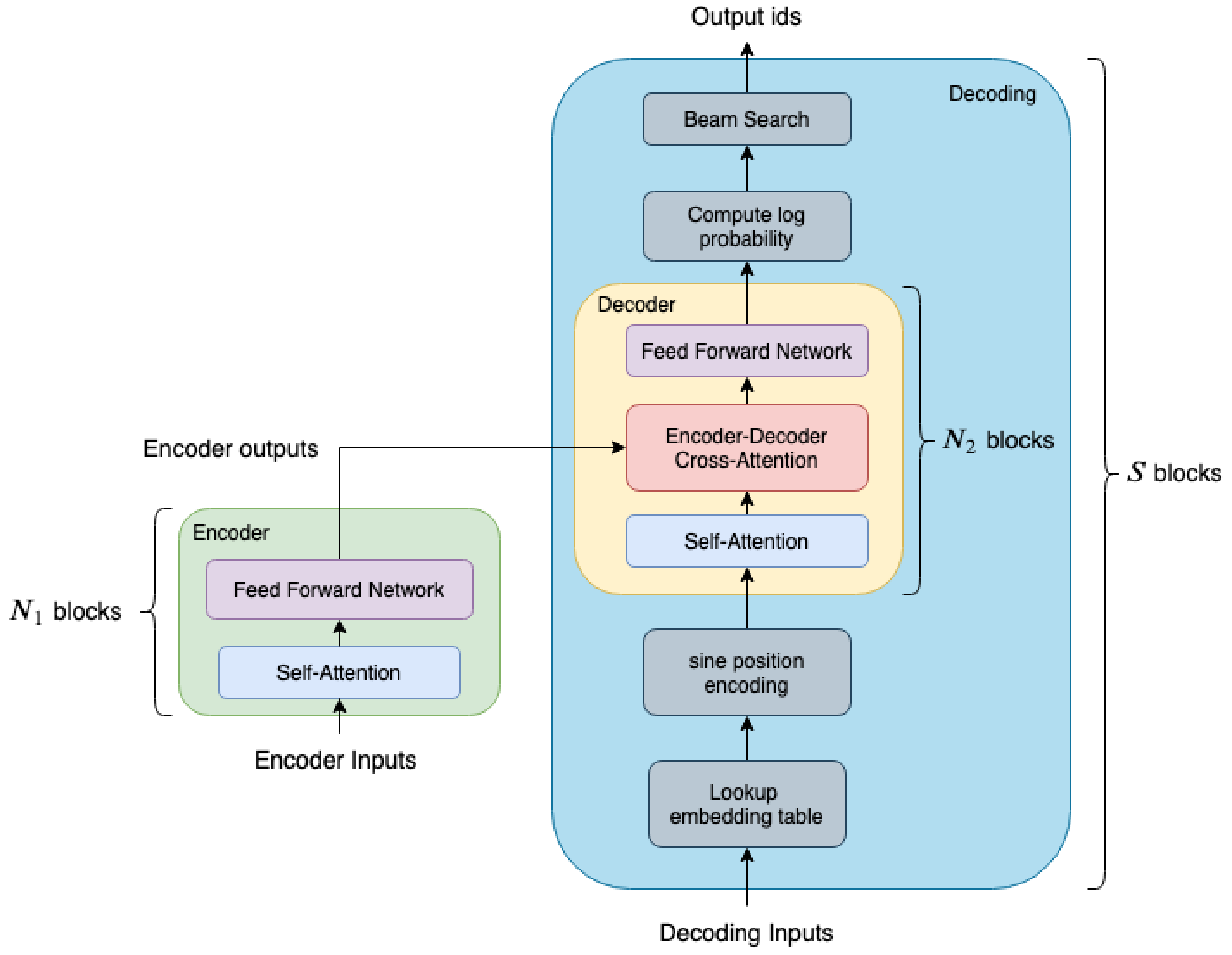
WHAT IS FASTER TRANSFORMER

FasterTransformer Architecture

Encoder: single layer BERT equivalented transformer

Decoder: single layer decoder transformer

Decoding: whole translating progress



WHAT IS FASTER TRANSFORMER

Decoder and Decoding

```
decoding(encoder_result, start_id):
    id = start_id
    while(finished == false):
        decoder_input = lookup_embedding_table(id)
        decoder_input = sine_position_encoding(decoder_input)
        for l = 1:num_layer:
                decoder_output = decoder(decoder_input, encoder_output, num_layer)
        prob = dense(decoder_output)
        id = beamsearch(prob, candidate_number)
```

WHAT IS FASTER TRANSFORMER

Summary

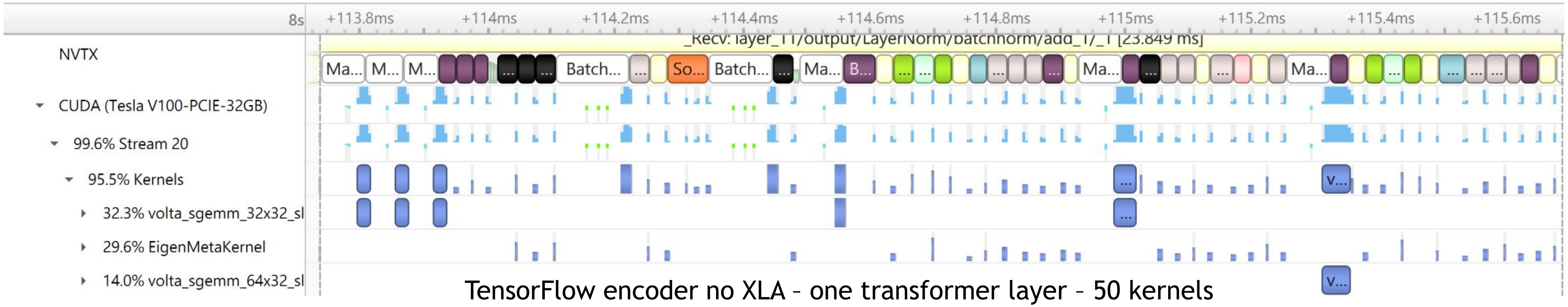
- FasterTransformer provides highly optimized transformer layer
    - Encoder transformer is based on BERT
    - Decoder transformer is based on OpenNMT-tf
    - Decoding contains whole process of translation, and it is also based on OpenNMT-tf
- Based on CUDA and cuBLAS
- Support both FP16 and FP32
- Provide C++ API and TensorFlow Op
- Source codes are available in https://github.com/NVIDIA/DeepLearningExamples/tree/master/FasterTransformer/v2

HOW TO OPTIMIZE

HOW TO OPTIMIZE

Where the bottleneck for Encoder?

- GPU is idle in many time
- Reason: kernels are too small -> kernel launch bound

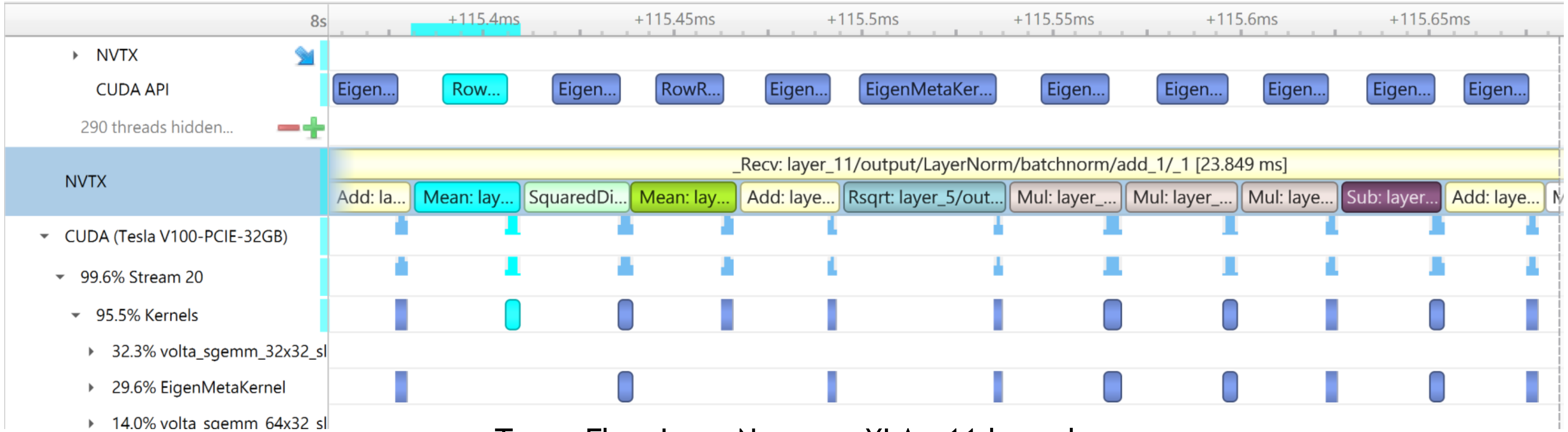

8s
+113.8ms
+114ms
+114.2ms
+114.4ms
+114.6ms
+114.8ms
+115ms
+115.2ms
+115.4ms
+115.6ms
NVTX
_Recv: layer_11/output/LayerNorm/batchnorm/add_1/_1 [23.849 ms]
CUDA (Tesla V100-PCIE-32GB)
99.6% Stream 20
95.5% Kernels
32.3% volta_sgemm_32x32_sl
29.6% EigenMetaKernel
14.0% volta_sgemm_64x32_sl

TensorFlow encoder no XLA – one transformer layer – 50 kernels
batch size 1, 12 heads, size per head 64, FP32

HOW TO OPTIMIZE

Where the bottleneck for Encoder?

- GPU is idle in many time
- Reason: kernels are too small
    - E.g., using 11 kernels (mean, add, …) to compute the LayerNorm

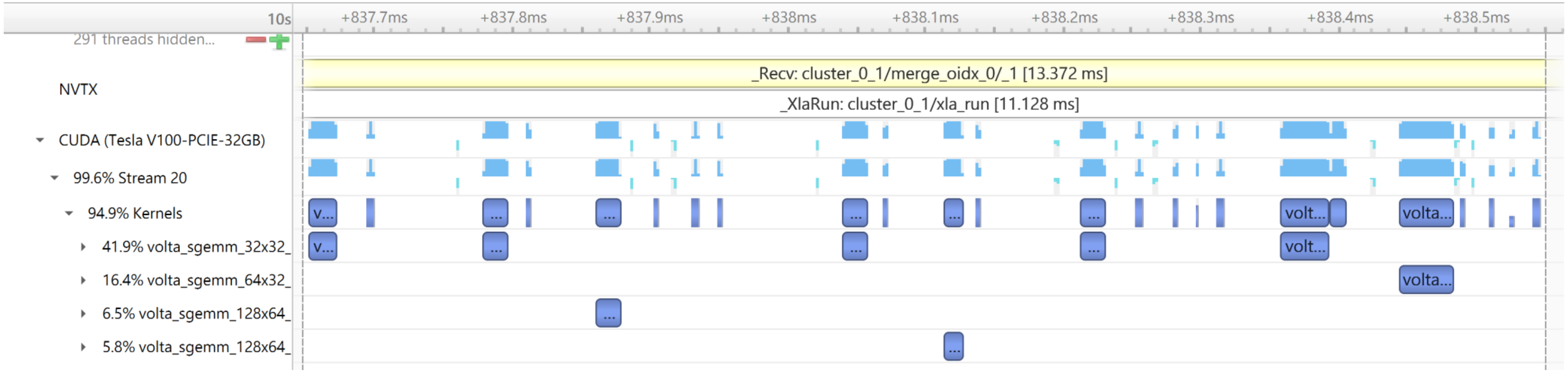


TensorFlow LayerNorm no XLA – 11 kernels
batch size 1, 12 heads, size per head 64, FP32

HOW TO OPTIMIZE

Where the bottleneck for Encoder?

- A simple solution: Using TensorFlow XLA to fuse kernel automatically
- Become better, but still many idle fraction

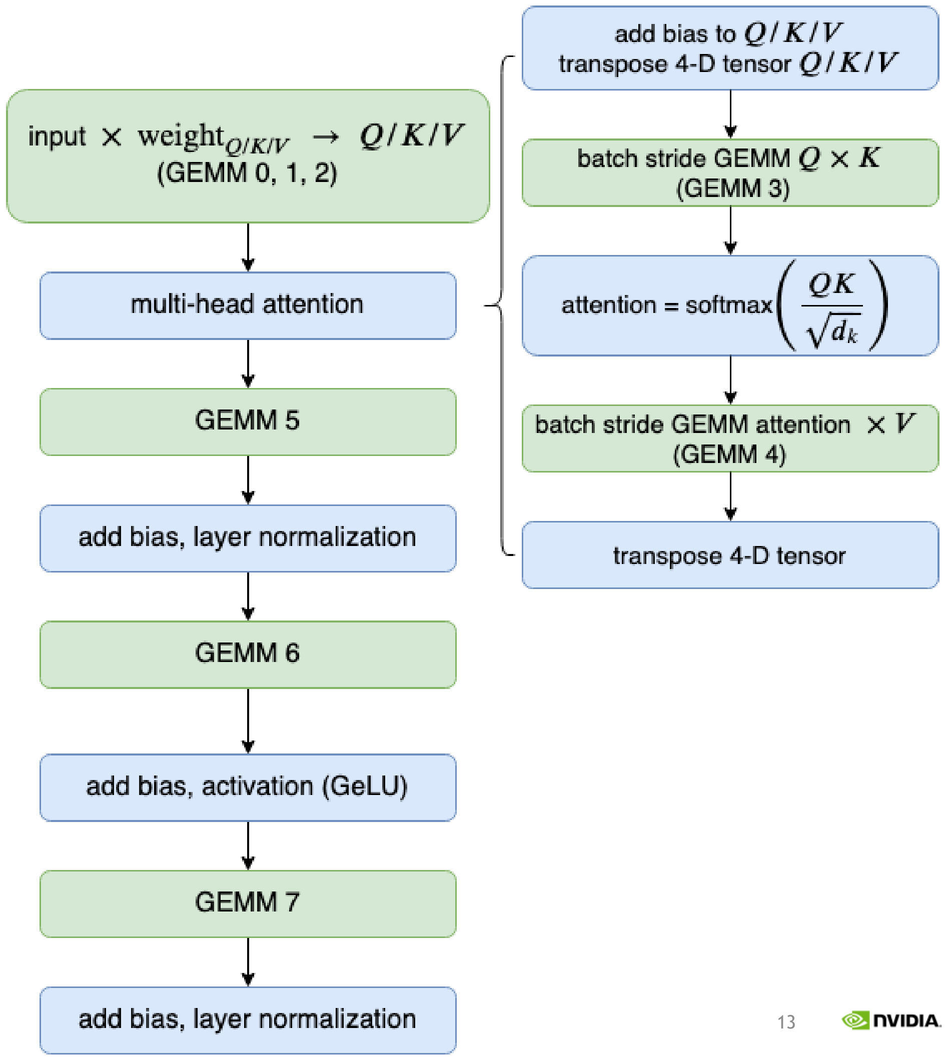


TensorFlow encoder with XLA – one transformer layer – 24 kernels
batch size 1, 12 heads, size per head 64, FP32

HOW TO OPTIMIZE

Fused Encoder

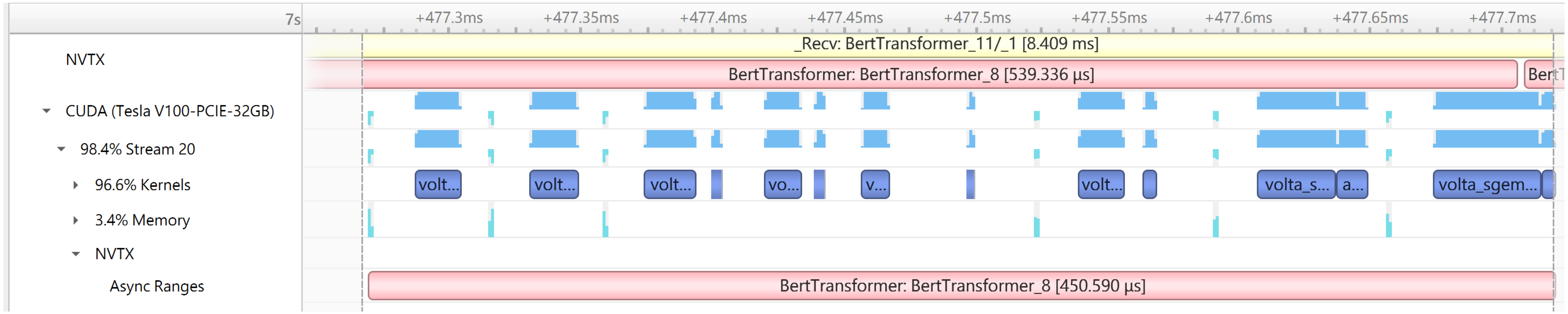
- GEMM (General Matrix Multiplication) is highly optimized by cuBLAS
- Fuse the kernels except GEMMs as much as possible
  - add bias + layer norm
  - add bias + activation
  - Transpose 3 matrices together in attention
  - Add bias + SoftMax
- 8 GEMMS and 6 custom kernels


input $\times$ weight$_{Q/K/V}$ $\rightarrow$ $Q/K/V$ (GEMM 0, 1, 2)
multi-head attention
GEMM 5
add bias, layer normalization
GEMM 6
add bias, activation (GeLU)
GEMM 7
add bias, layer normalization
add bias to $Q/K/V$ transpose 4-D tensor $Q/K/V$
batch stride GEMM $Q \times K$ (GEMM 3)
attention = softmax$\left(\frac{QK}{\sqrt{d_k}}\right)$
batch stride GEMM attention $\times V$ (GEMM 4)
transpose 4-D tensor

HOW TO OPTIMIZE

Fused Encoder

- Timeline after fusion

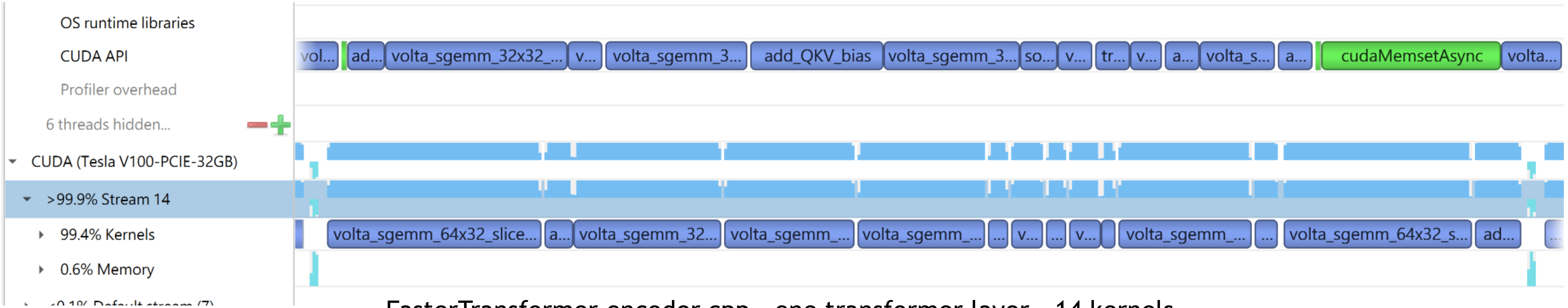


FasterTransformer encoder op – one transformer layer – 14 kernels
batch size 1, 12 heads, size per head 64, FP32

HOW TO OPTIMIZE

Fused Encoder

- Timeline after fusion
- The pure cpp API can provide better performance

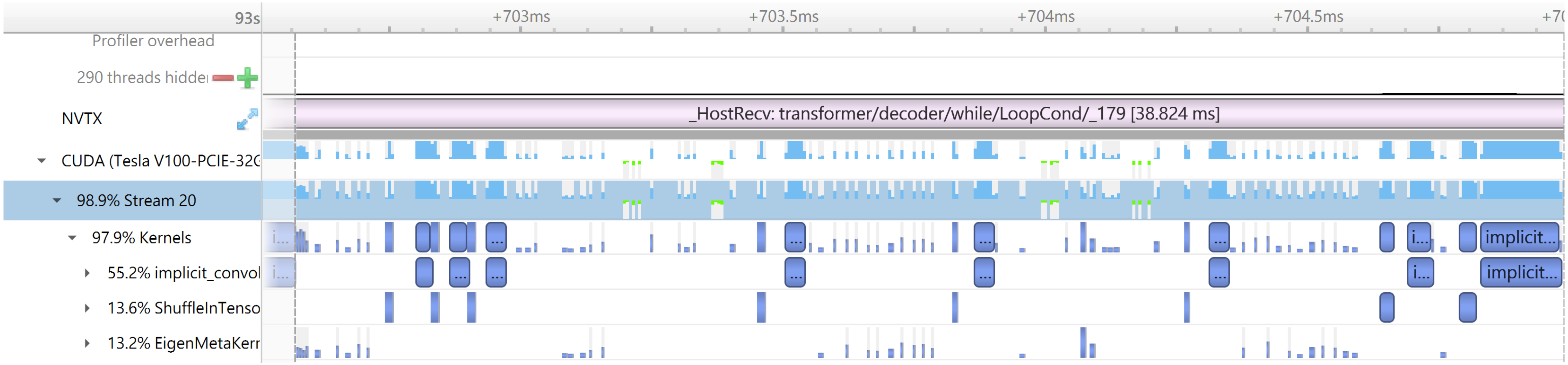


FasterTransformer encoder cpp – one transformer layer – 14 kernels
batch size 1, 12 heads, size per head 64, FP32

HOW TO OPTIMIZE

Where the bottleneck for Decoder?

- Decoder has similar, but more small kernels



TensorFlow Decoder no XLA – one transformer layer – about 70 kernels
batch size 1, 8 heads, size per head 64, FP32

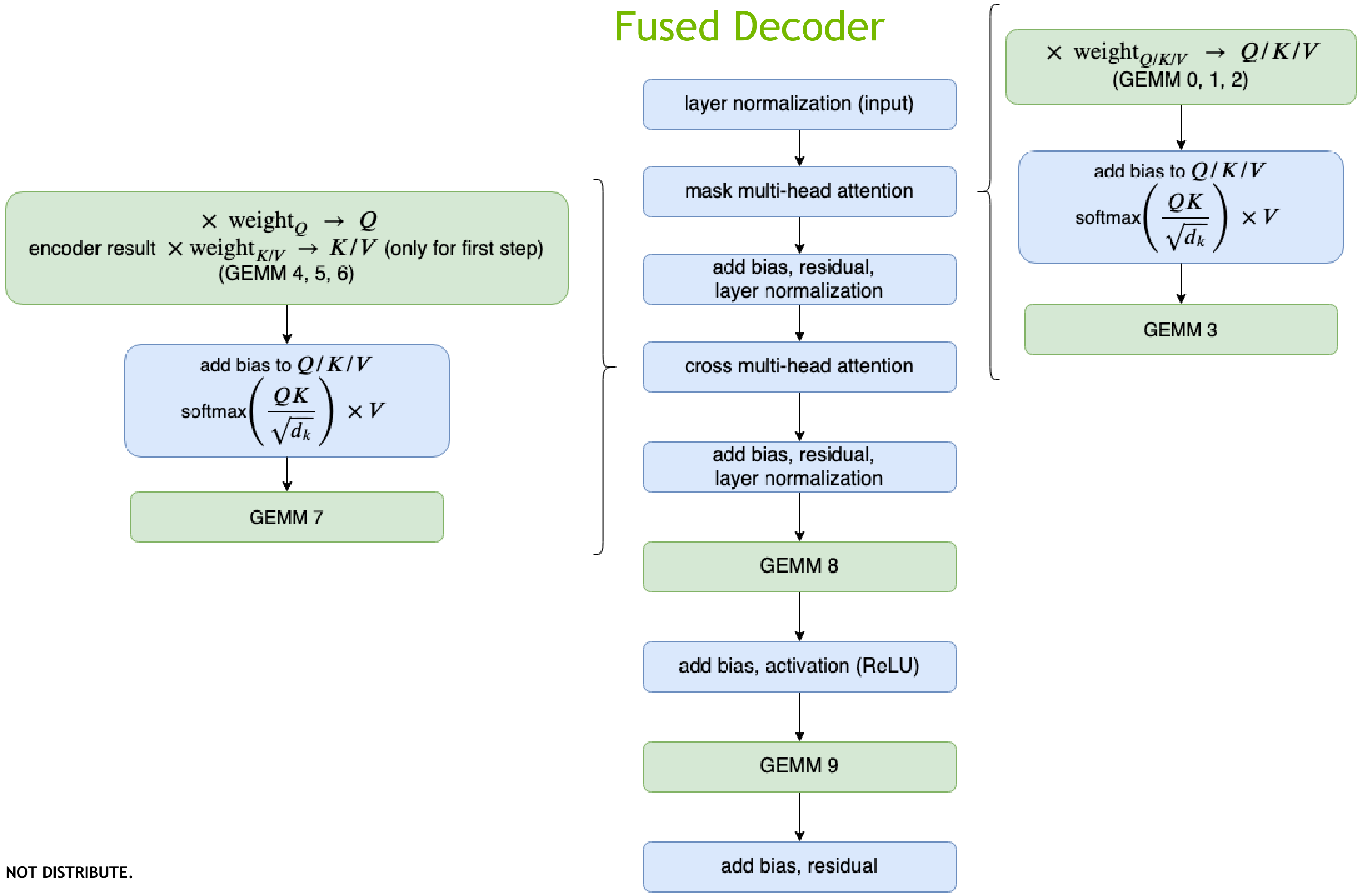
HOW TO OPTIMIZE

Fused Decoder

- We have tested the TensorFlow XLA for Decoder too, but the speed is not improved.
- Fuse the kernels except GEMMs as much as possible
    - Fuse the multi-head attention into one kernel (add bias, transpose, batch multiplication)
- FasterTransformer compresses the kernel number from about 70 kernels to 16 kernels

HOW TO OPTIMIZE

Fused Decoder

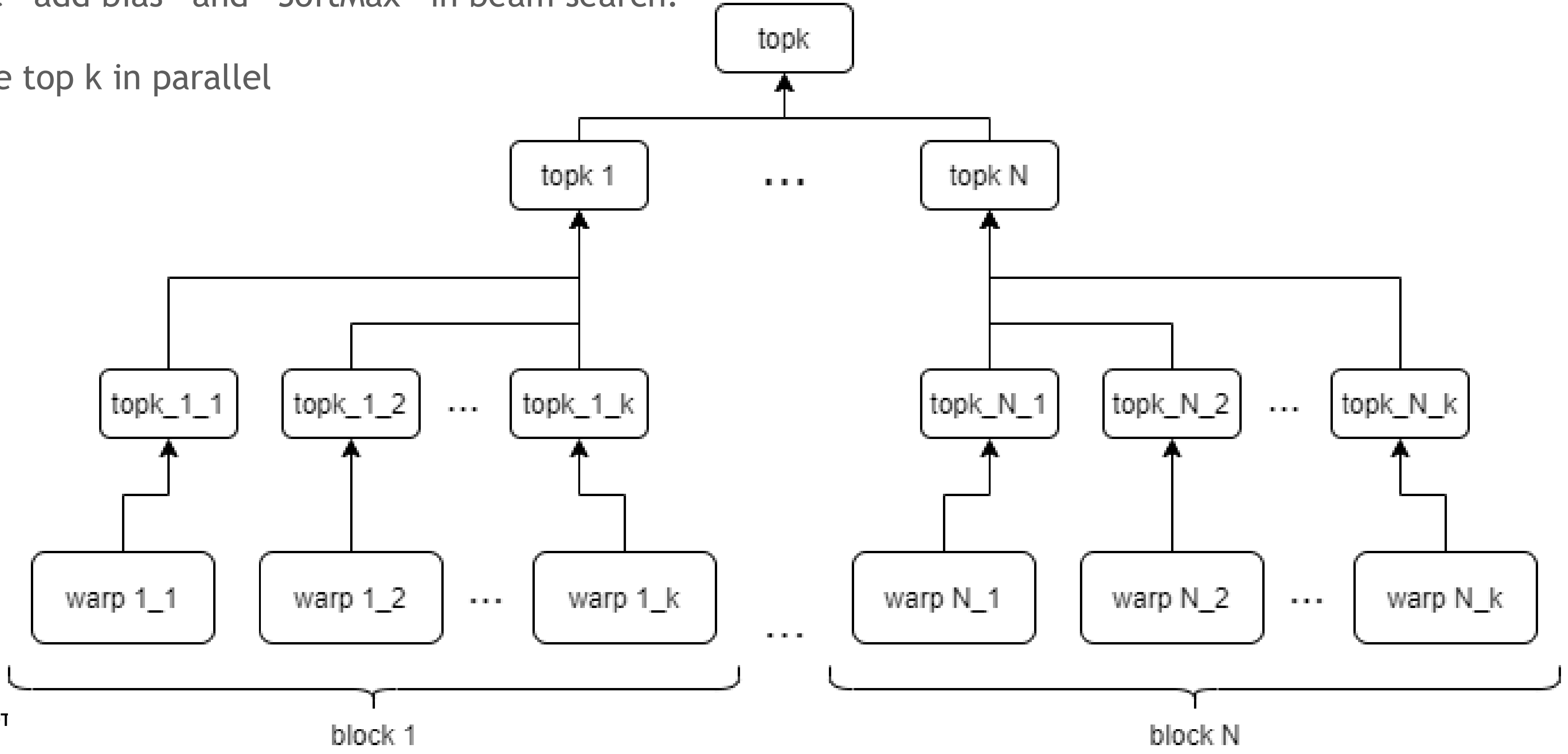


layer normalization (input)
mask multi-head attention
add bias, residual, layer normalization
cross multi-head attention
add bias, residual, layer normalization
GEMM 8
add bias, activation (ReLU)
GEMM 9
add bias, residual
$\times$ weight$_{Q/K/V}$ $\rightarrow$ $Q/K/V$ (GEMM 0, 1, 2)
add bias to $Q/K/V$
softmax$\left(\frac{QK}{\sqrt{d_k}}\right) \times V$
GEMM 3
$\times$ weight$_Q$ $\rightarrow$ Q
encoder result $\times$ weight$_{K/V}$ $\rightarrow$ K/V (only for first step)
(GEMM 4, 5, 6)
add bias to $Q/K/V$
softmax$\left(\frac{QK}{\sqrt{d_k}}\right) \times V$
GEMM 7

HOW TO OPTIMIZE

Decoding kernel

- Fuse the “add bias” and “SoftMax” in beam search.
- Compute top k in parallel



HOW TO OPTIMIZE

Further optimization

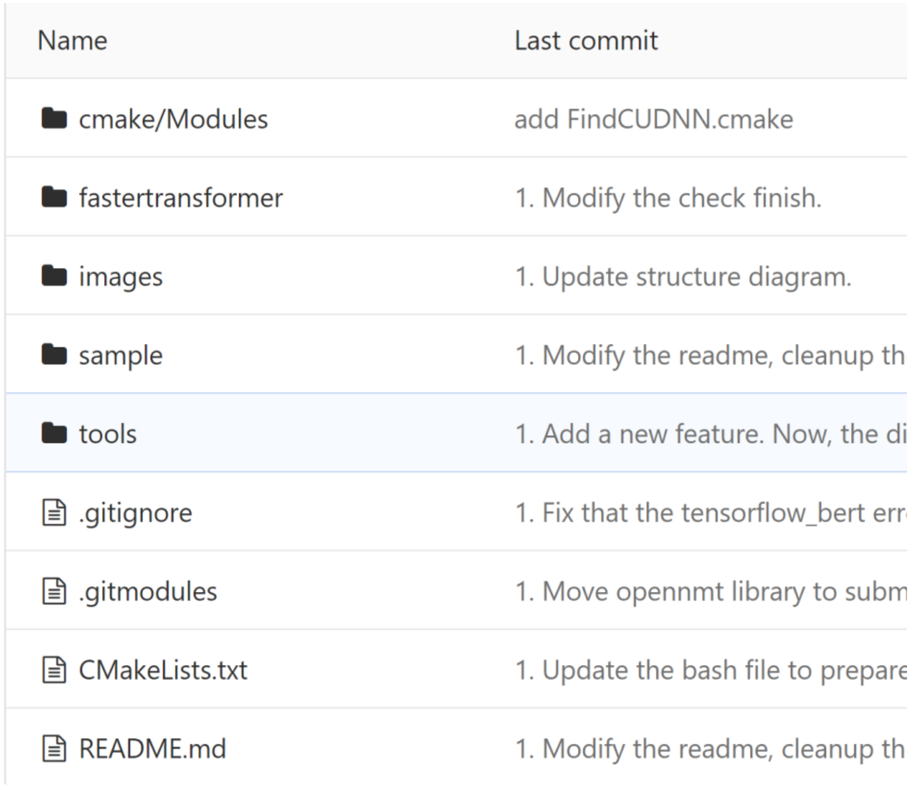
- Use half2 instruction to reduce the number of memory read/write in FP16
- Use lower precision but faster function, like replace “expf” by “__expf”
- Use different implementation and block/grid numbers for FP32 and FP16 for some kernel like SoftMax
- Provide a tool to check all possible GEMM configuration and choose the best one
- Use __shfl_xor_sync to accelerate the reduced operations

HOW TO USE

FASTER TRANSFORMER

Code Organization - overall

- fastertransformer: source codes
- sample: C++ and TensorFlow sample codes
- tools: choose the best config for GEMM.
- README.md: demonstrate how to use and the performance of FasterTransformer

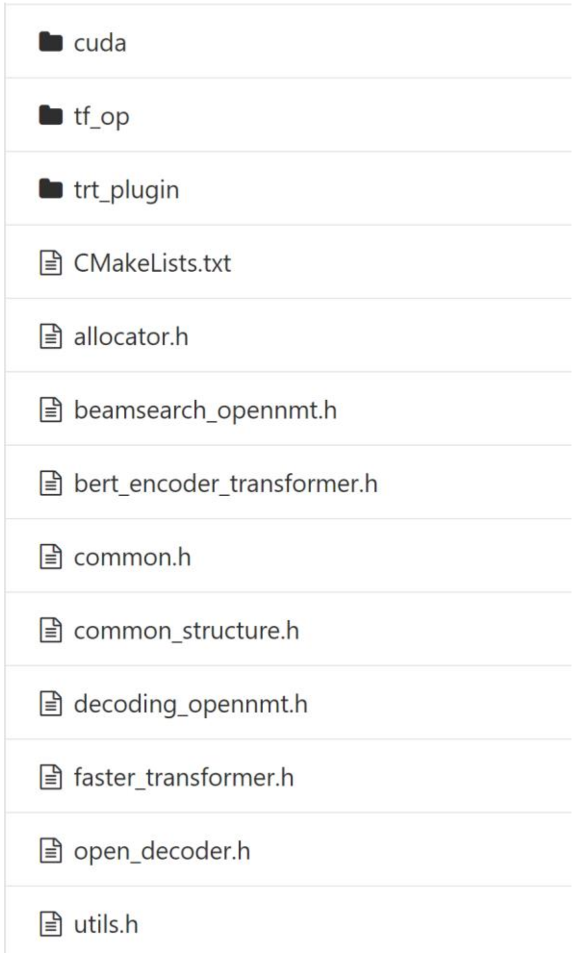


Name	Last commit
cmake/Modules	add FindCUDNN.cmake
fastertransformer	1. Modify the check finish.
images	1. Update structure diagram.
sample	1. Modify the readme, cleanup th
tools	1. Add a new feature. Now, the di
.gitignore	1. Fix that the tensorflow_bert err
.gitmodules	1. Move opennmt library to subm
CMakeLists.txt	1. Update the bash file to prepare
README.md	1. Modify the readme, cleanup th

FASTER TRANSFORMER

Code Organization - fastertransformer

- fastertransformer
  - bert_encoder_transformer.h: encoder transformer layer
  - open_decoder.h: decoder transformer layer
  - beamsearch_opennmt.h: beam search
  - decoding_opennmt.h: decoding
  - cuda: cuda kernels for encoder, decoder and decoding

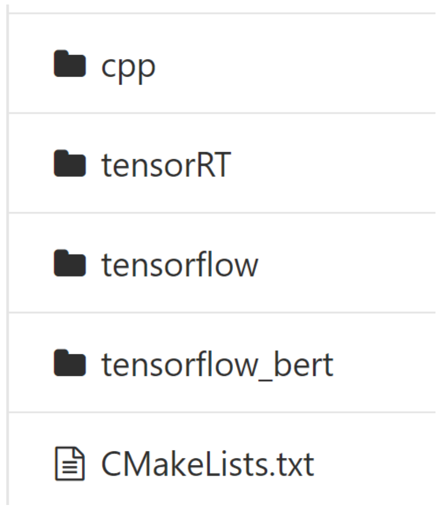


```
cuda
tf_op
trt_plugin
CMakeLists.txt
allocator.h
beamsearch_opennmt.h
bert_encoder_transformer.h
common.h
common_structure.h
decoding_opennmt.h
faster_transformer.h
open_decoder.h
utils.h
```

FASTER TRANSFORMER

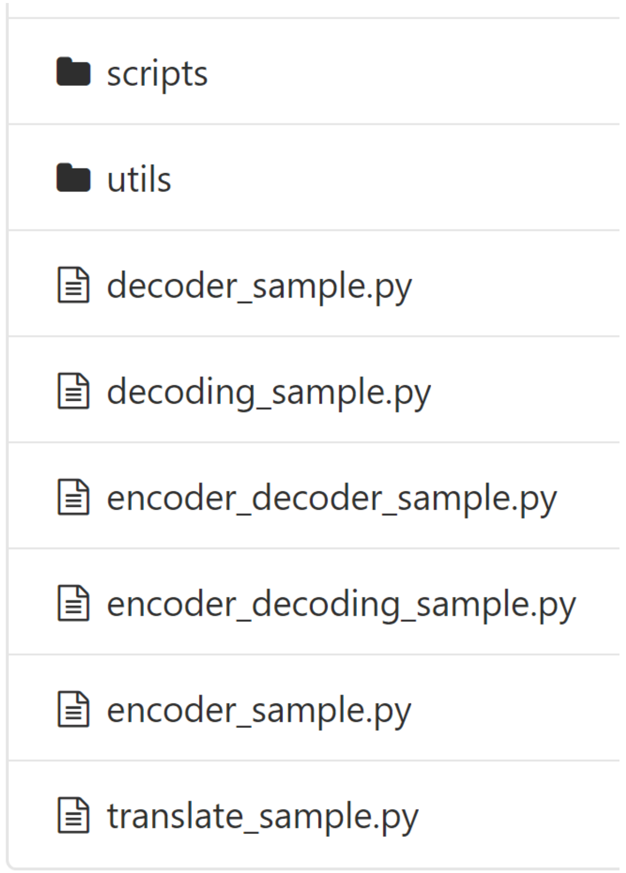
Code Organization - sample

- sample
  - tensorflow: TensorFlow Op samples for encoder, decoder and decoding
  - cpp: c++ interface samples for encoder and decoding



cpp
tensorRT
tensorflow
tensorflow_bert
CMakeLists.txt

sample



scripts
utils
decoder_sample.py
decoding_sample.py
encoder_decoder_sample.py
encoder_decoding_sample.py
encoder_sample.py
translate_sample.py

Sample/tensorflow

FASTER TRANSFORMER

How to use in C++

- Create encoder/decoder/decoding object by hyper-parameters

```
BertEncoderTransformer<EncoderTraits_> *encoder_transformer_ =
        new BertEncoderTransformer<EncoderTraits_>(allocator,
                                                   batch_size,
                                                   from_seq_len,
                                                   to_seq_len,
                                                   head_num,
                                                   size_per_head);
```

- Set the weights to encoder/decoder/decoding params

```
EncoderInitParam<T> encoder_param; //init param here

encoder_param.from_tensor = d_from_tensor;
encoder_param.to_tensor = d_from_tensor;
encoder_param.self_attention.query_weight.kernel = d_attr_kernel_Q;
encoder_param.self_attention.key_weight.kernel = d_attr_kernel_K;
```

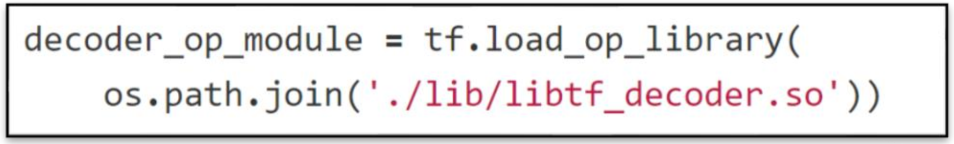
- Forward by the params

```
for (int i = 0; i < ite; ++i)
  encoder_transformer_->forward();
```

FASTER TRANSFORMER

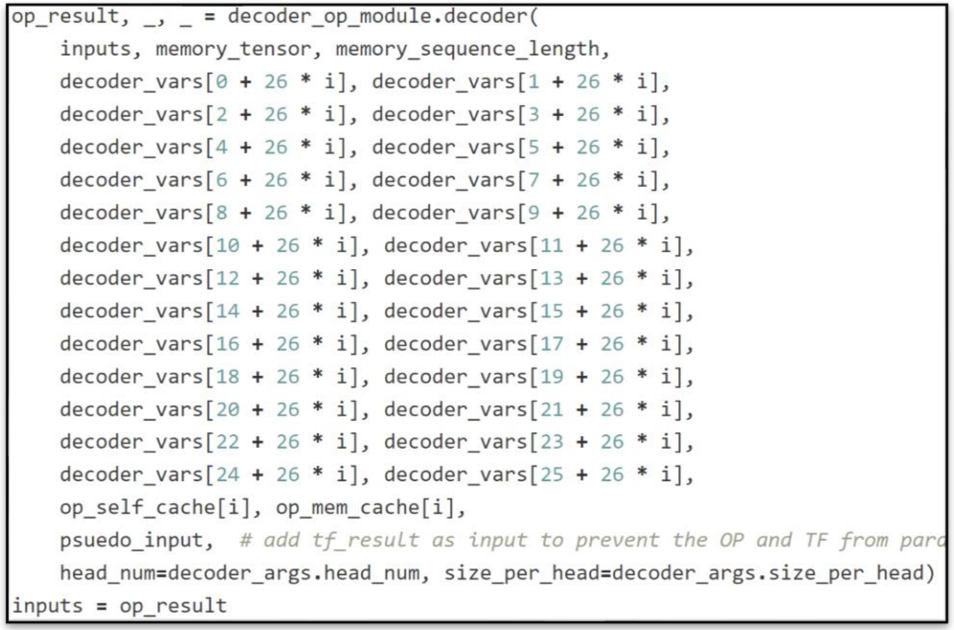
How to use in TensorFlow

- Load the shared objects



```
decoder_op_module = tf.load_op_library(
    os.path.join('./lib/libtf_decoder.so'))
```

- Set the weights to encoder/decoder/decoding params



```
op_result, _, _ = decoder_op_module.decoder(
    inputs, memory_tensor, memory_sequence_length,
    decoder_vars[0 + 26 * i], decoder_vars[1 + 26 * i],
    decoder_vars[2 + 26 * i], decoder_vars[3 + 26 * i],
    decoder_vars[4 + 26 * i], decoder_vars[5 + 26 * i],
    decoder_vars[6 + 26 * i], decoder_vars[7 + 26 * i],
    decoder_vars[8 + 26 * i], decoder_vars[9 + 26 * i],
    decoder_vars[10 + 26 * i], decoder_vars[11 + 26 * i],
    decoder_vars[12 + 26 * i], decoder_vars[13 + 26 * i],
    decoder_vars[14 + 26 * i], decoder_vars[15 + 26 * i],
    decoder_vars[16 + 26 * i], decoder_vars[17 + 26 * i],
    decoder_vars[18 + 26 * i], decoder_vars[19 + 26 * i],
    decoder_vars[20 + 26 * i], decoder_vars[21 + 26 * i],
    decoder_vars[22 + 26 * i], decoder_vars[23 + 26 * i],
    decoder_vars[24 + 26 * i], decoder_vars[25 + 26 * i],
    op_self_cache[i], op_mem_cache[i],
    psuedo_input,  # add tf_result as input to prevent the OP and TF from para
    head_num=decoder_args.head_num, size_per_head=decoder_args.size_per_head)
inputs = op_result
```

- Session run

PERFORMANCE

FASTERTRANSFORMER PERFORMANCE

Environment Setting

- image: nvcr.io/nvidia/tensorflow:19.07-py2
    - CUDA 10.1
    - TensorFlow 1.14
    - Python 2.7
- CPU: Intel(R) Xeon(R) Gold 6132 CPU @ 2.60GHz
- NVIDIA Tesla T4 (with mclk 5000MHz, pclk 1590MHz)
- NVIDIA Tesla V100 (with mclk 877MHz, pclk 1380MHz)

FASTERTRANSFORMER PERFORMANCE

Encoder benchmark on NVIDIA Tesla V100

Batch size	TensorFlow XLA (ms)	FasterTransformer (ms)	speedup
100	13.96	9.57	1.459
200	26.47	18.37	1.440
300	38.40	27.41	1.401
400	49.65	35.63	1.393
500	62.20	44.57	1.396

12 layers, 32 sequence length, 12 heads, 64 size per head (BERT base), under FP16

FASTERTRANSFORMER PERFORMANCE

benchmark on NVIDIA Tesla T4

Sequence length	TF (ms)	Decoder (ms)	Decoder speedup	Decoding (ms)	Decoding speedup
32	463.90	135.70	3.42	55.34	8.38
64	917.90	267.55	3.43	110.38	8.31
128	1789.45	527.75	3.40	241.82	7.40

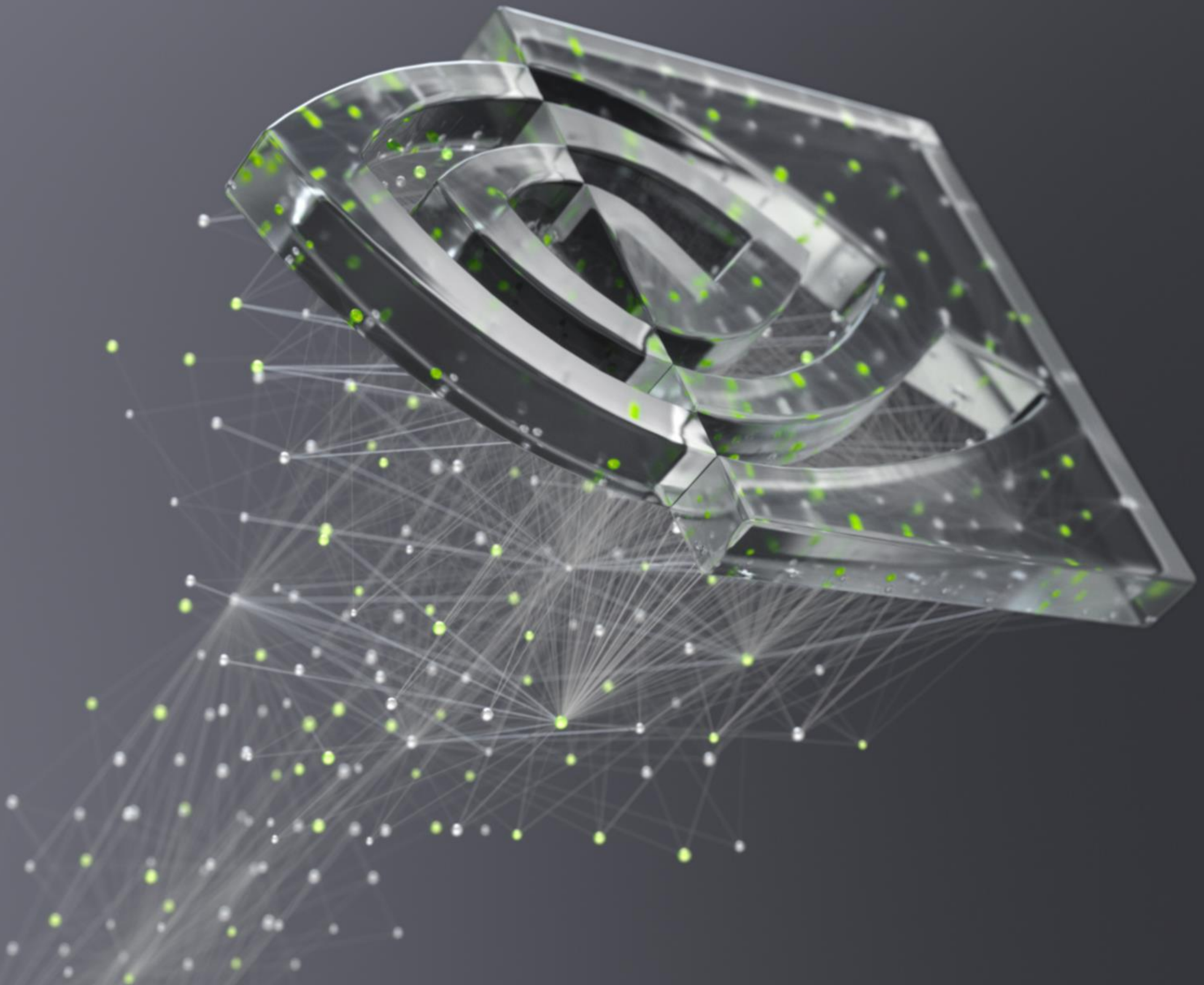
Batch size 1, beam width 4, 8 heads, 64 size per head, 6 layers, vocabulary size 30000, FP32
TF: TensorFlow; Decoder: FasterTransformer decoder; Decoding: FasterTransformer decoding

FASTERTRANSFORMER PERFORMANCE

benchmark on NVIDIA Tesla T4

Data type	TF (ms)	Decoder (ms)	Decoder speedup	Decoding (ms)	Decoding speedup
FP32	2234.10	1439.81	1.55	1248.67	1.79
FP16	1601.06	814.50	1.96	693.14	2.31

Batch size 256, sequence length 32, beam width 4, 8 heads, 64 size per head, 6 layers, vocabulary size 30000
TF: TensorFlow; Decoder: FasterTransformer decoder; Decoding: FasterTransformer decoding


NVIDIA