



GPU PROFILING 기법을 통한 DEEP LEARNING 성능 최적화 기법 소개

Gwangsoo Hong, Solution Architect, ghong@nvidia.com



AGENDA

Introduction to Nsight Systems
Profiling from CLI
NVTX (NVIDIA Tools Extension)
Deep learning Optimization
Getting Started Resources



INTRODUCTION TO NSIGHT SYSTEMS

HOW TO SPEED-UP NETWORK

- ▶ Use the latest GPU and more GPU?
- ▶ Wait NVIDIA to release the new GPU or newer library?
- ▶ **Optimize Neural Network Computing or Something**
 - ▶ Need to understand your network operation

We are talking about this

PROFILING NEURAL NETWORK

Profiling with cProfiler + Snakeviz



Soumith Chintala ✅

@soumithchintala

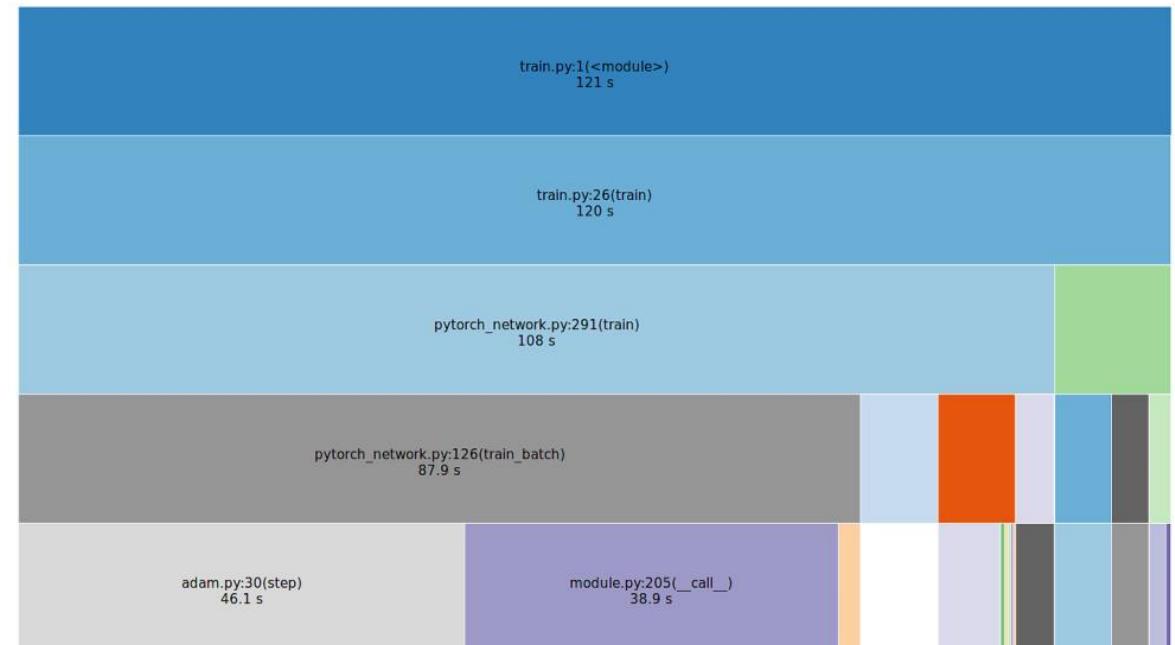
Following

Replying to @sleepinyourhat @PyTorch

I've found this helpful when profiling pytorch code: jiffyclub.github.io/snakeviz/
It takes cProfile outputs and gives much nicer viz

7:01 AM - 27 May 2017

5 Retweets 46 Likes

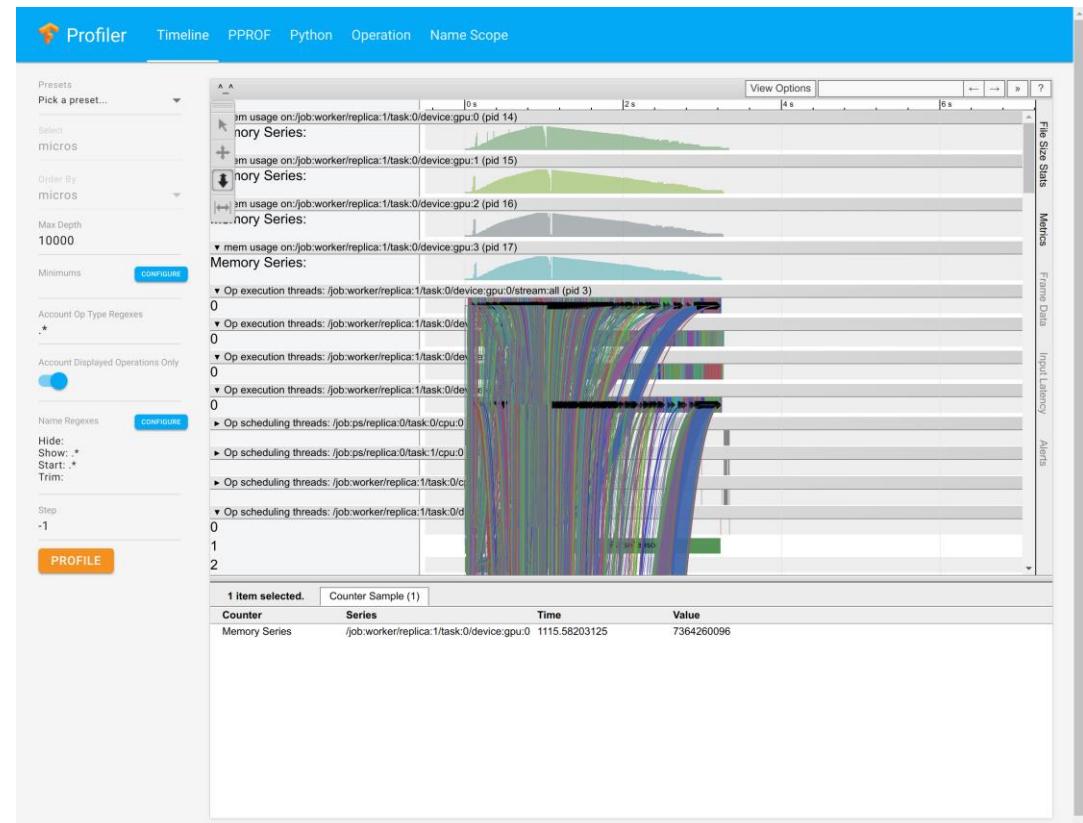


```
$python -m cProfile -o 100_percent_gpu_utilization.prof train.py  
$snakeviz 100_percent_gpu_utilization.prof
```

TENSORFLOW PROFILER

<https://github.com/tensorflow/profiler-ui>

- ▶ Show timeline and can trace network
- ▶ Still difficult to understand



NVIDIA PROFILING TOOLS

Nsight Systems



Nsight Compute



Nsight Visual Studio Edition

Nsight Eclipse Edition



NVIDIA Profiler (nvprof)

NVIDIA Visual Profiler (nvvprof)



CUPTI (CUDA Profiling Tools Interface)

NSIGHT PRODUCT FAMILY

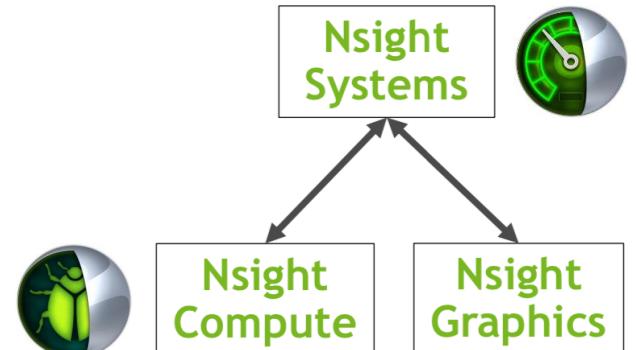
- ▶ Standalone Performance Tools

- ▶ Nsight Systems system-wide application algorithm tuning
- ▶ Nsight Compute Debug/optimize specific CUDA kernel
- ▶ Nsight Graphics Debug/optimize specific graphics

- ▶ IDE plugins

- ▶ Nsight Eclipse Edition/Visual Studio editor, debugger, some perf analysis

Workflow

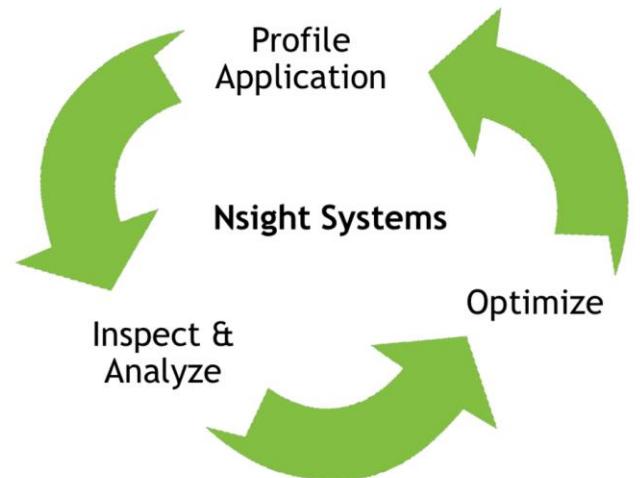


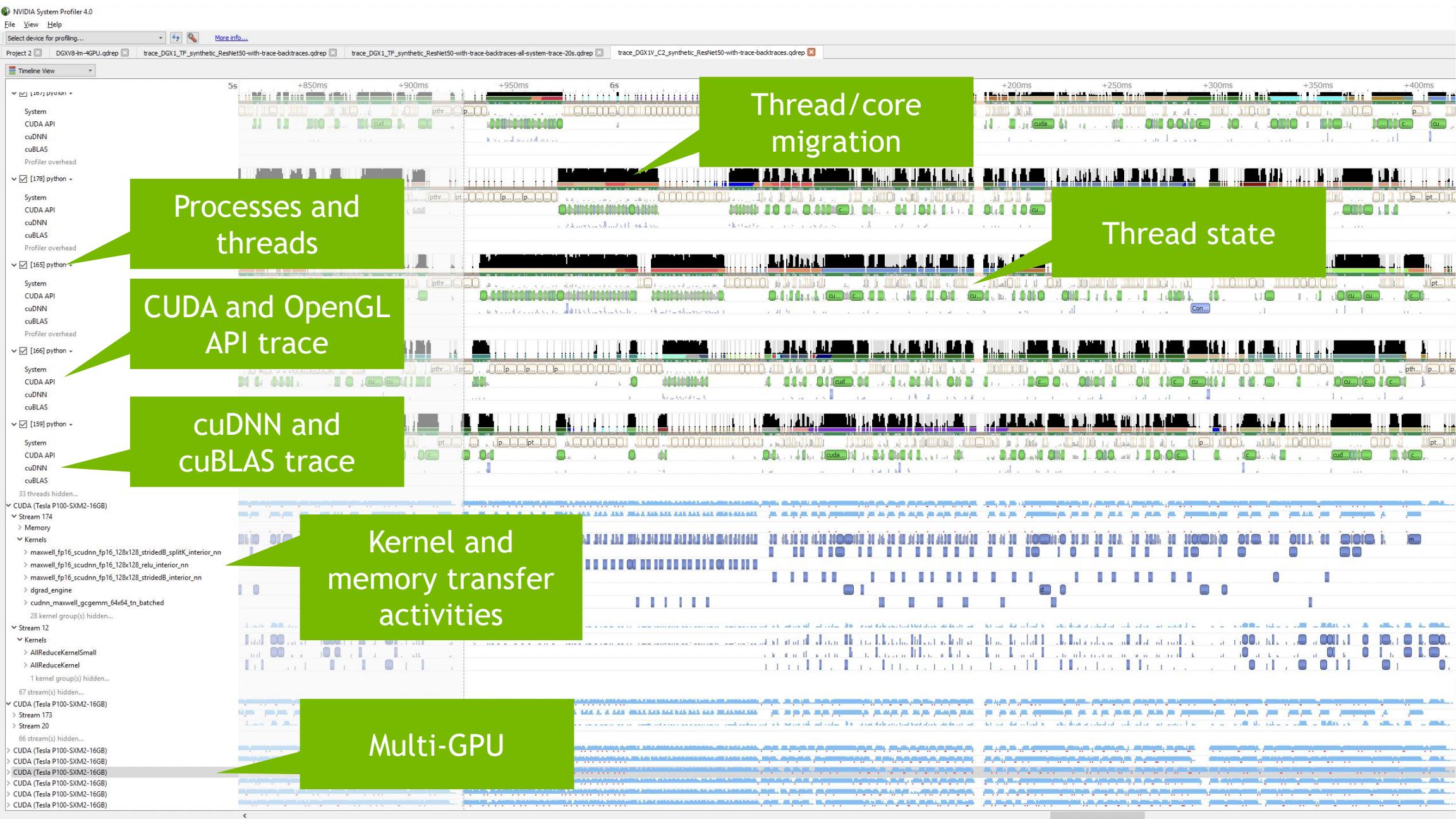


NSIGHT SYSTEMS

Overview

- ▶ Profile **System-wide** application
 - ▶ Multi-process tree, GPU workload trace, etc
- ▶ Investigate your workload across multiple CPUs and GPUs
 - ▶ CPU algorithms, utilization, and thread states
 - ▶ GPU streams kernels, memory transfers, etc
 - ▶ NVTX, CUDA & Library API, etc
- ▶ Ready for Big Data
 - ▶ docker, user privilege (linux), cli, etc





CPU (80)

Threads (78)

[1221583] Caffe2F

NVTX

CUDA API

NVTX Tracing

[1221583] Caffe2F

NVTX

CUDA API

Weight... ConvGradient [1...

cudaEventSynchronize

[1221589] Caffe2F

NVTX

CUDA API

Spati...

cud... cudaEv...

[1221587] Caffe2F

[1221582] Caffe2F

73 threads hidden...

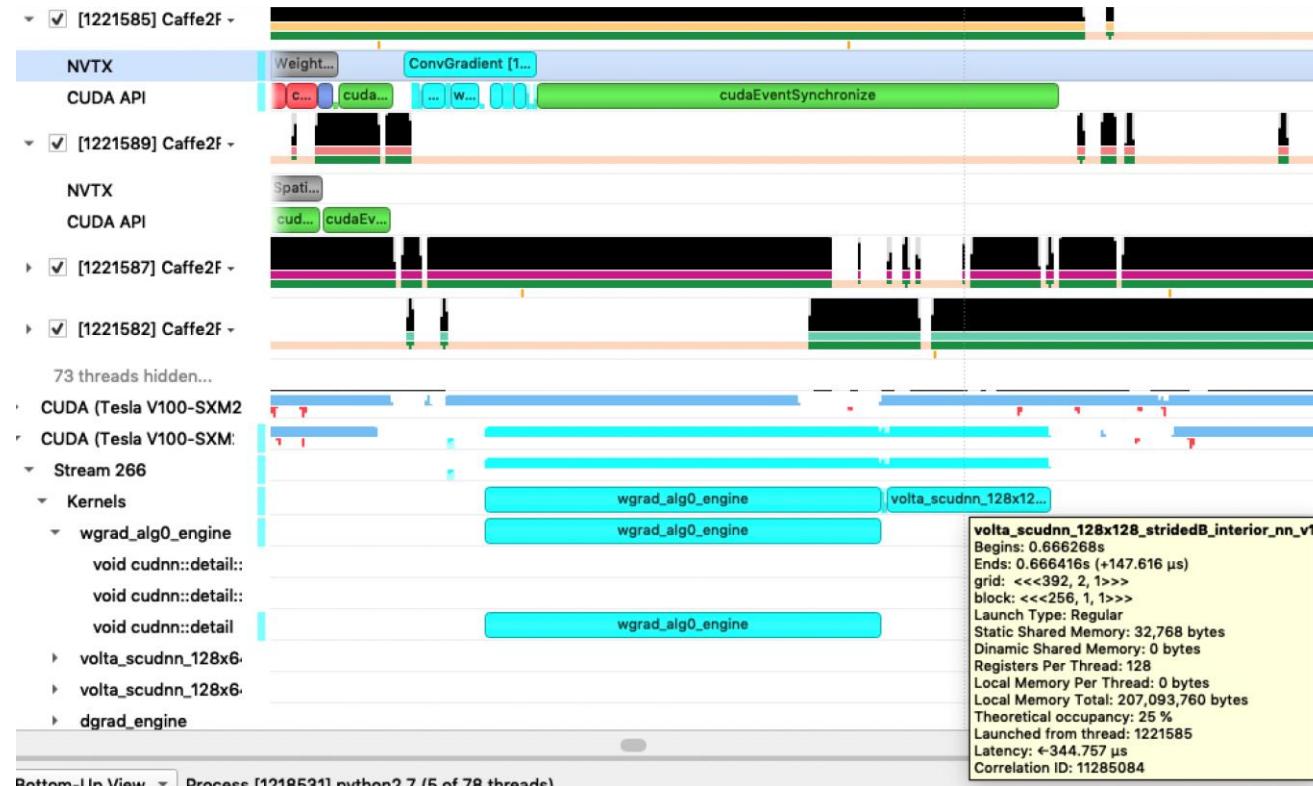
CUDA (Tesla V100-SXM2

CUDA (Tesla V100-S)

CUDA Kernel running
Time: 0.666129s

TRANSITIONING TO PROFILE A KERNEL

Dive into kernel analysis



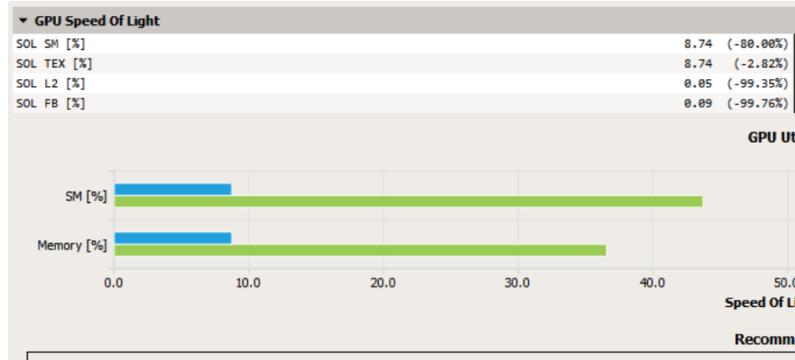


NVIDIA NSIGHT COMPUTE

Next Generation Kernel Profiler

- ▶ Interactive CUDA API debugging and kernel profiling
- ▶ Fast Data Collection
- ▶ Graphical and multiple kernel comparison reports
- ▶ Improved Workflow and Fully Customizable (Baselining, Programmable UI/Rules)
- ▶ Command Line, Standalone, IDE Integration
- ▶ Platform Support
 - ▶ OS: Linux(x86,ARM), Windows, OSX (host only)
 - ▶ GPUs: Pascal, Volta, Turing

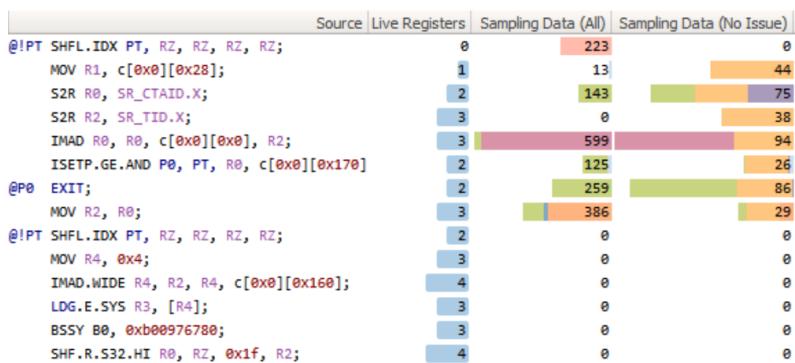
Kernel Profile Comparisons with Baseline



Metric Data

inst_executed [inst]	16,528.00	16,528.00	-	13,476.00	13,476.00	-
l1tex_sol_pct [%]			14.33			n/a
launch_block_size			128.00			128.00
launch_function_pcs			47,611,587,968.00			12,273,728.00
launch_grid_size			4,132.00			3,369.00
launch_occupancy_limit_blocks [block]			32.00			32.00
launch_occupancy_limit_registers [register]			21.00			21.00
launch_occupancy_limit_shared_mem [bytes]			384.00			384.00
launch_occupancy_limit_warp [warps]			16.00			16.00
launch_occupancy_per_block_size			3,638.00			3,638.00
launch_occupancy_per_register_count			5,792.00			5,792.00
launch_occupancy_per_shared_mem_size			2,260.00			2,260.00
launch_registers_per_thread [register/thread]			17.00			17.00
launch_shared_mem_config_size [bytes]			49,152.00			49,152.00
launch_shared_mem_per_block_dynamic [bytes/block]			0.00			0.00
launch_shared_mem_per_block_static [bytes/block]			20.00			20.00
launch_thread_count [thread]			528,896.00			431,232.00
ltc_sol_pct [%]			3.23			42.11
memory_access_size_type [bytes]			2.00; 32.00; 32.00; 32.00; 2.00; 32.00; 32.00; 32.00			6.93

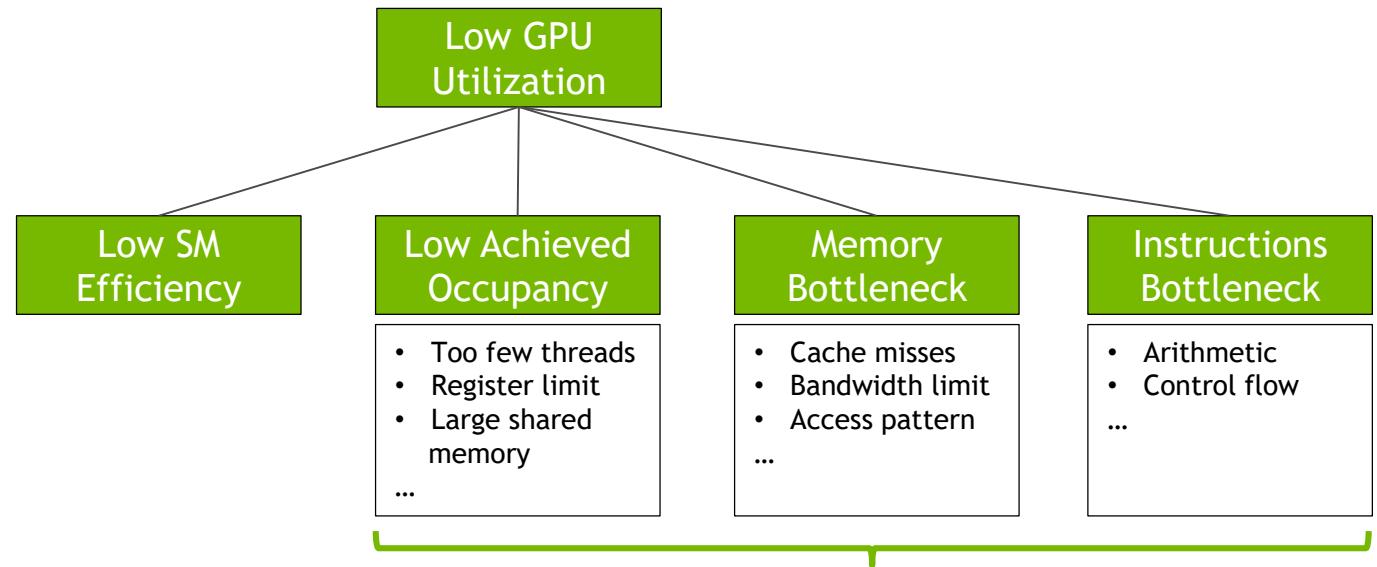
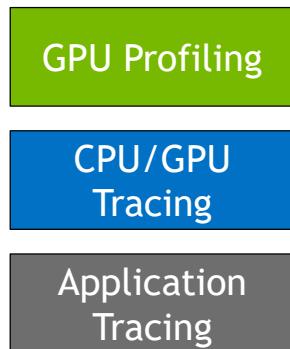
Source Correlation



PROFILING GPU APPLICATION

Focusing GPU Computing

How to measure



NVIDIA (Visual) Profiler / Nsight Compute

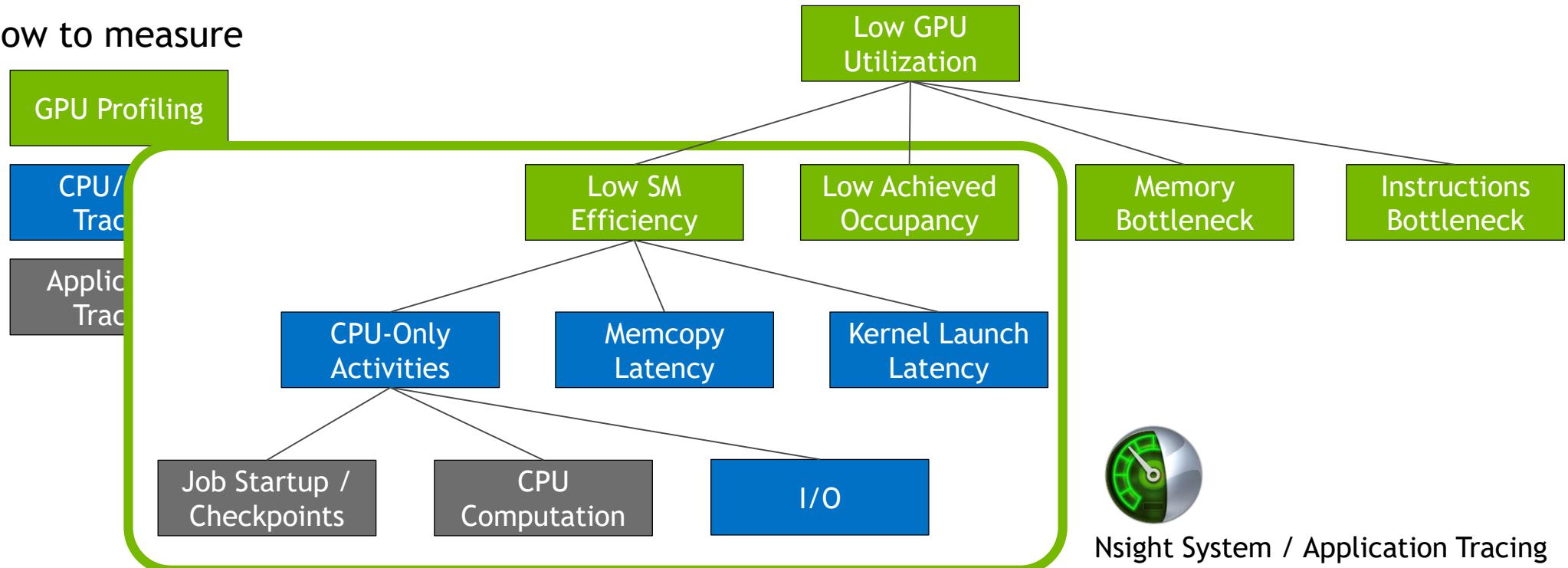


NVIDIA Supports them with cuDNN, cuBLAS, and so on

PROFILING GPU APPLICATION

Focusing System Operation

How to measure





HOW TO USE

NSIGHT SYSTEMS PROFILE

Profile with CLI

APIs to be traced

```
$ nsys profile -t cuda,osrt,nvtx,cudnn,cublas \
-o baseline.qdstrm -w true python main.py
```

Name of output file Show output on console Application command

Automatic conversion of .qdstrm temp results file to .qdrep format if converter utility is available.

cuda - GPU kernel

osrt - OS runtime

nvtx - NVIDIA Tools Extension

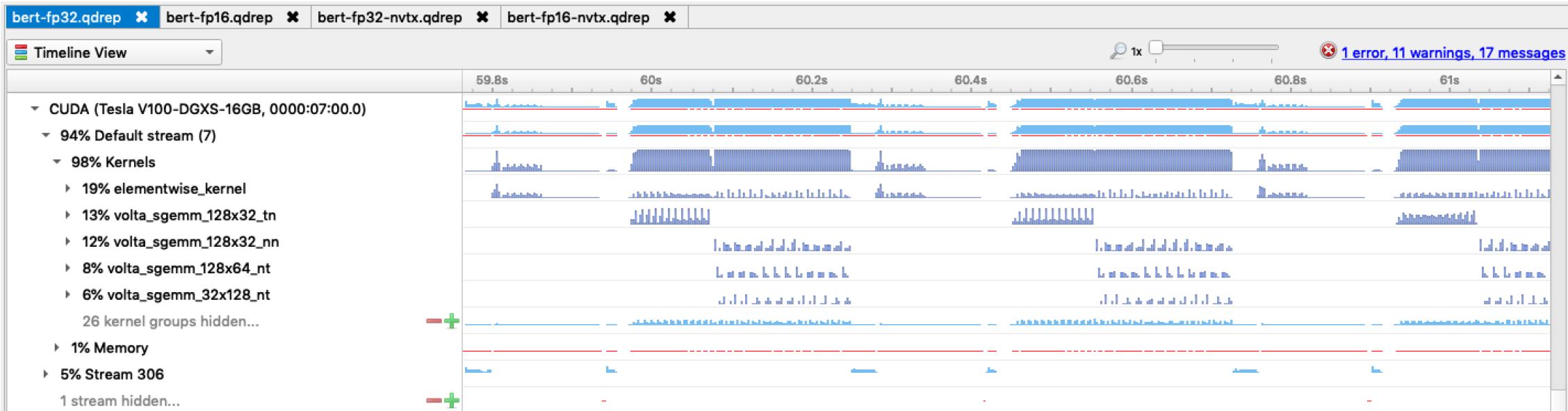
cudnn - CUDA Deep NN library

cublas - CUDA BLAS library

https://docs.nvidia.com/nsight-systems/#nsight_systems/2019.3.6-x86/06-cli-profiling.htm

NSIGHT SYSTEMS PROFILE

No NVTX



- ▶ Difficult to understand → no useful

NVTX (NVIDIA TOOLS EXTENSION)

NVTX ANNOTATIONS

```
def train(args, model, device, train_loader, optimizer, epoch):
    model.train()
    for batch_idx, (data, target) in enumerate(train_loader):

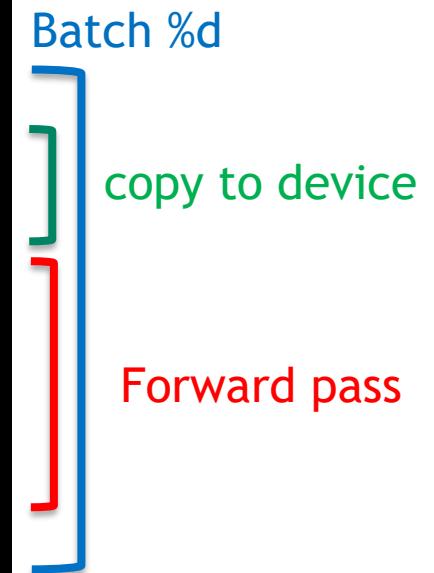
        data, target = data.to(device), target.to(device)

        optimizer.zero_grad()
        output = model(data)
        loss = F.nll_loss(output, target)
```

NVTX ANNOTATIONS

NVTX in PyTorch

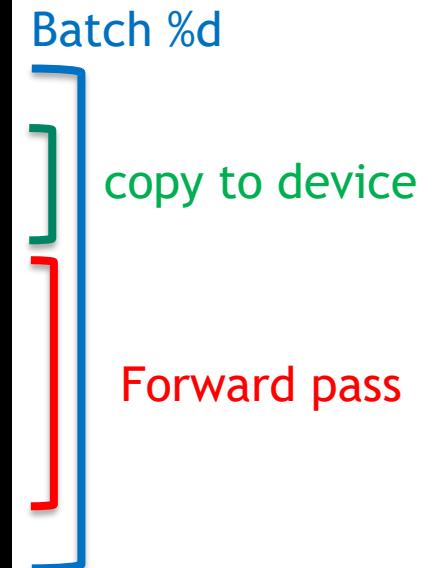
```
import torch.cuda.nvtx as nvtx
def train(args, model, device, train_loader, optimizer, epoch):
    model.train()
    for batch_idx, (data, target) in enumerate(train_loader):
        nvtx.range_push("Batch " + str(batch_idx))
        nvtx.range_push("Copy to device")
        data, target = data.to(device), target.to(device)
        nvtx.range_pop()
        nvtx.range_push("Forward pass")
        optimizer.zero_grad()
        output = model(data)
        loss = F.nll_loss(output, target)
        nvtx.range_pop()
        nvtx.range_pop()
```



NVTX ANNOTATIONS

NVTX using cupy pakage

```
from cupy.cuda import nvtx
def train(args, model, device, train_loader, optimizer, epoch):
    model.train()
    for batch_idx, (data, target) in enumerate(train_loader):
        nvtx.RangePush("Batch " + str(batch_idx))
        nvtx.RangePush("Copy to device")
        data, target = data.to(device), target.to(device)
        nvtx.RangePop()
        nvtx.RangePush("Forward pass")
        optimizer.zero_grad()
        output = model(data)
        loss = F.nll_loss(output, target)
        nvtx.RangePop()
        nvtx.RangePop()
```



NSIGHT SYSTEM PROFILE

NVTX range marker tip

- ▶ NVTX for data loading (data augmentation)

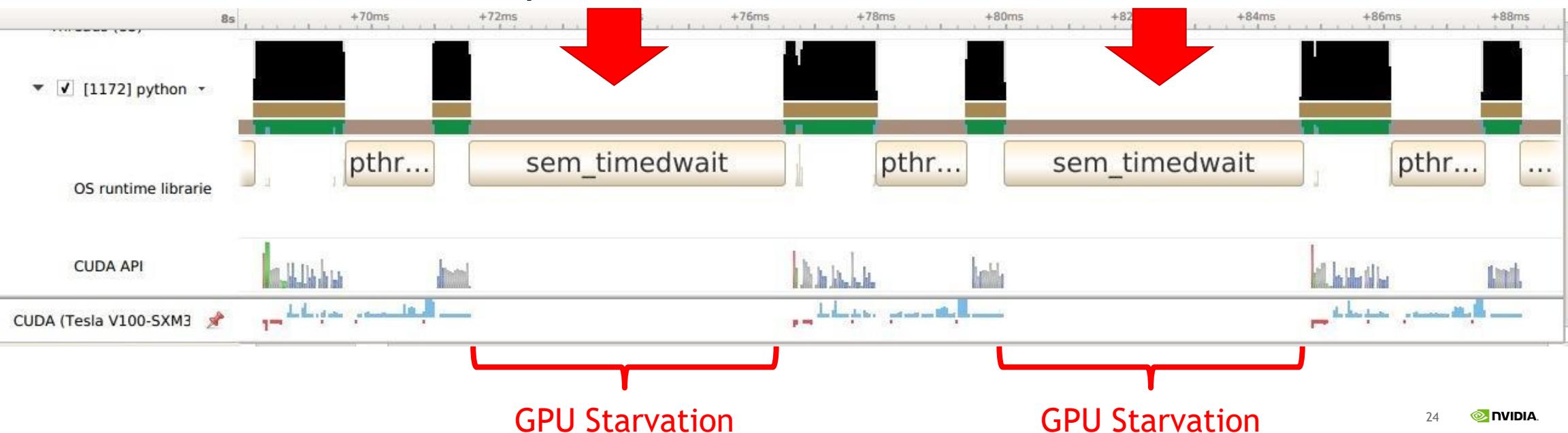
```
for batch_idx, (data,target) in enumerate(train_loader):
```



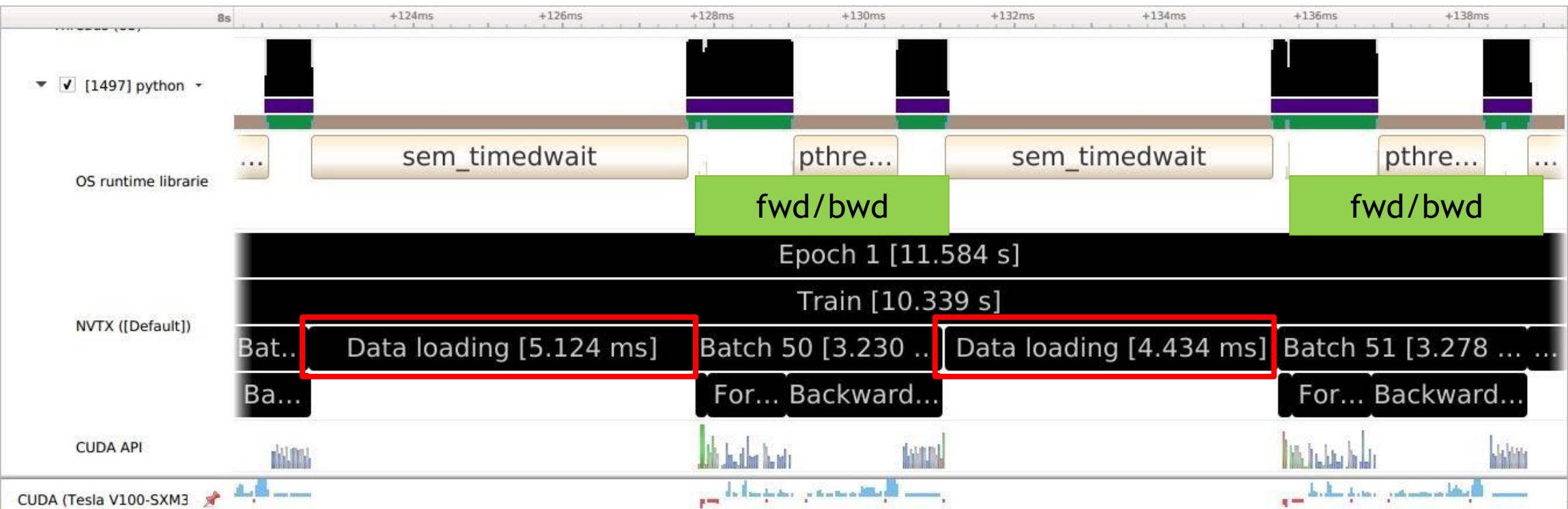
```
nvtx.range_push('Data loading')
(data,target) = train_loader.next()
nvtx.range_pop()
```

BASELINE PROFILE

- ▶ MNIST Training: 89 sec, <5% utilization
- ▶ CPU waits on a semaphore and starves the GPU!



BASELINE PROFILE (WITH NVTX)



- ▶ GPU is idle during **data loading**
- ▶ Data is loaded using a single thread. This starves the GPU!

5.1ms

OPTIMIZE SOURCE CODE

- ▶ Data loader was configured to use 1 worker thread

```
kwargs = {'num_workers': 1, 'pin_memory': True if use_cuda else {}}
```

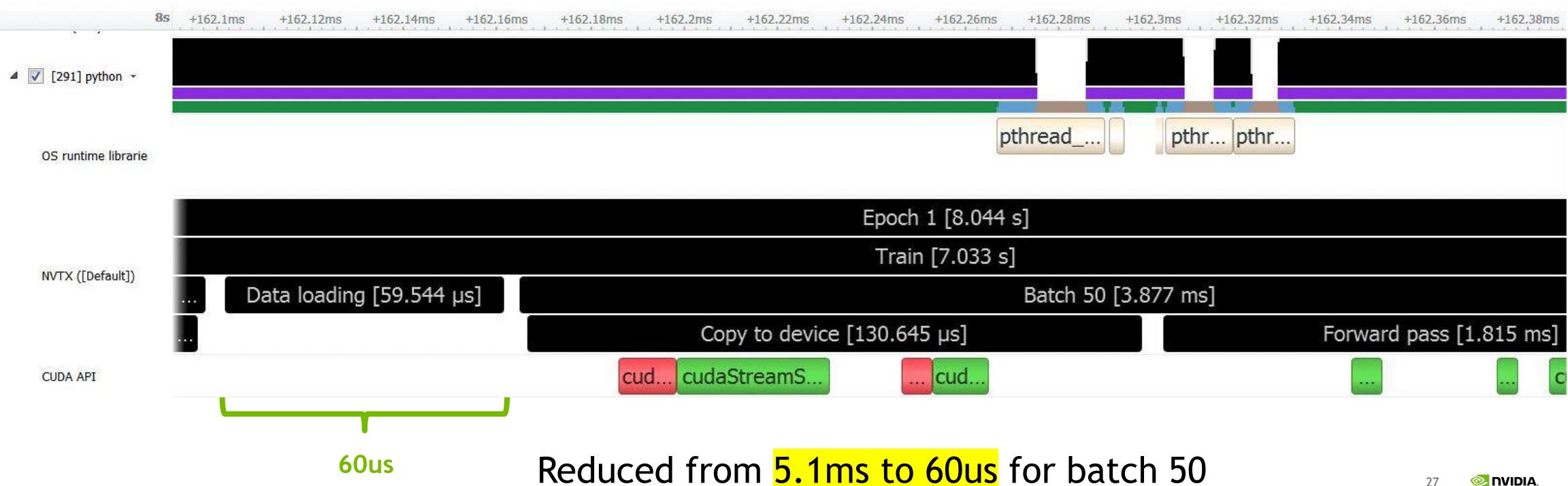


- ▶ Let's switch to using 8 worker threads:

```
kwargs = {'num_workers': 8, 'pin_memory': True if use_cuda else {}}
```

AFTER OPTIMIZATION

- ▶ Time for data loading reduced for each bath





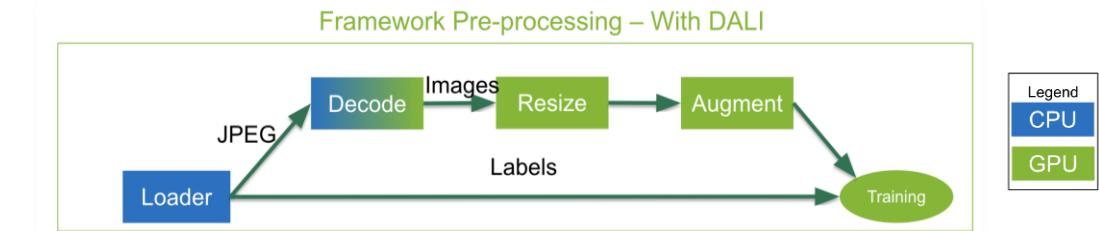
DEEP LEARNING OPTIMIZATION

OPTIMIZATION STRATEGY FOR DL

- ▶ Algorithm optimization
 - ▶ Tensor Cores
- ▶ Data pipeline optimization
 - ▶ DALI (Data loading Library)

$$D = \begin{pmatrix} A_{0,0} & A_{0,1} & A_{0,2} & A_{0,3} \\ A_{1,0} & A_{1,1} & A_{1,2} & A_{1,3} \\ A_{2,0} & A_{2,1} & A_{2,2} & A_{2,3} \\ A_{3,0} & A_{3,1} & A_{3,2} & A_{3,3} \end{pmatrix}_{\text{FP16 or FP32}} \begin{pmatrix} B_{0,0} & B_{0,1} & B_{0,2} & B_{0,3} \\ B_{1,0} & B_{1,1} & B_{1,2} & B_{1,3} \\ B_{2,0} & B_{2,1} & B_{2,2} & B_{2,3} \\ B_{3,0} & B_{3,1} & B_{3,2} & B_{3,3} \end{pmatrix}_{\text{FP16}} + \begin{pmatrix} C_{0,0} & C_{0,1} & C_{0,2} & C_{0,3} \\ C_{1,0} & C_{1,1} & C_{1,2} & C_{1,3} \\ C_{2,0} & C_{2,1} & C_{2,2} & C_{2,3} \\ C_{3,0} & C_{3,1} & C_{3,2} & C_{3,3} \end{pmatrix}_{\text{FP16 or FP32}}$$

- ▶ Available in **Volta** and **Turing** architecture GPUs
- ▶ 125 Tflops in FP16 vs. 15.7 Tflops in FP32 (**8x** speed-up)
- ▶ Optimized **4x4x4** dot operation (GEMM)





TRAINING OPTIMIZATION CASE

NVTX TAGGING

BERT in PyTorch

```
loss = model(input_ids, segment_ids, input_mask, start_positions, end_positions)

...
if args.fp16:
    optimizer.backward(loss)
else:
    loss.backward()
if (step + 1) % args.gradient_accumulation_steps == 0:
    if args.fp16:
        # modify Learning rate with special warm up BERT uses
        # if args.fp16 is False, BertAdam is used and handles this automatically
        lr_this_step = args.learning_rate * warmup_linear.get_lr(global_step, args.warmup_proportion)
        for param_group in optimizer.param_groups:
            param_group['lr'] = lr_this_step
    optimizer.step()
    optimizer.zero_grad()
    global_step += 1
```

NVTX TAGGING

BERT in PyTorch

```
nvtx.range_push("Batch " + str(step))
nvtx.range_push("Forward pass")
loss = model(input_ids, segment_ids, input_mask, start_positions, end_positions)
nvtx.range_pop()

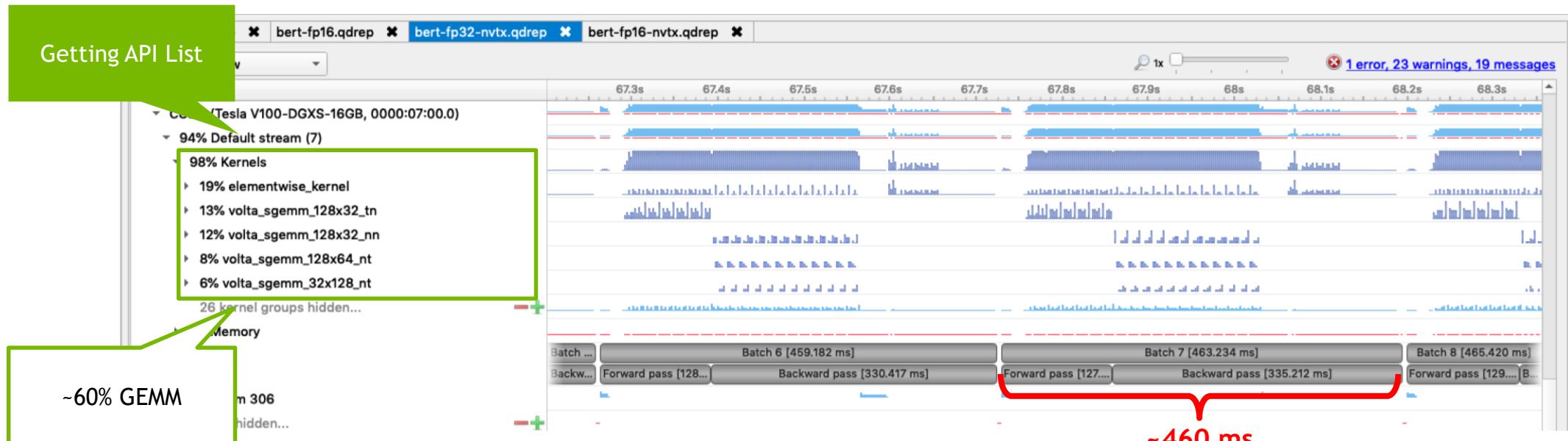
...
nvtx.range_push("Backward pass")
if args.fp16:
    optimizer.backward(loss)
else:
    loss.backward()
if (step + 1) % args.gradient_accumulation_steps == 0:
if args.fp16:
    # modify Learning rate with special warm up BERT uses
    # if args.fp16 is False, BertAdam is used and handles this automatically
    lr_this_step = args.learning_rate * warmup_linear.get_lr(global_step, args.warmup_proportion)
    for param_group in optimizer.param_groups:
        param_group['lr'] = lr_this_step
optimizer.step()
optimizer.zero_grad()
global_step += 1
nvtx.range_pop()
nvtx.range_pop()
```

The diagram illustrates the use of NVTX range tags in a PyTorch BERT training loop. It shows three nested ranges:

- A green bracket labeled "forward pass" covers the range from "Batch" to "Backward pass".
- A red bracket labeled "backward pass" covers the range from "Backward pass" to "optimizer.step()".
- A blue bracket labeled "Batch %d" covers the entire range from "Batch" to "nvtx.range_pop()".

ALGORITHM OPTIMIZATION - TRAINING

Single Precision (FP32) Training for BERT



ALGORITHM OPTIMIZATION - TRAINING

Automatic Mixed Precision (FP32 + FP16) Training for BERT

- 2.1x Speed up (~460ms vs. ~222ms)

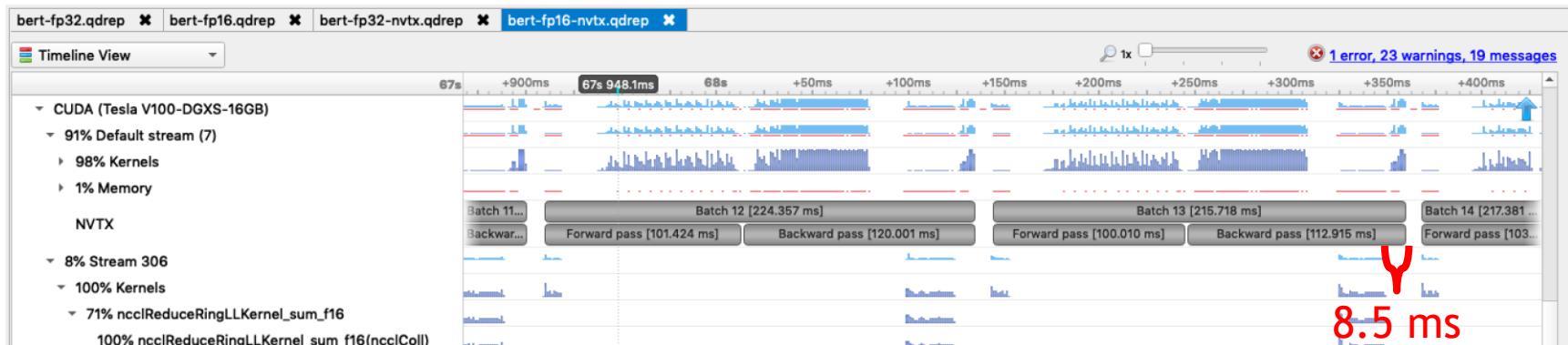
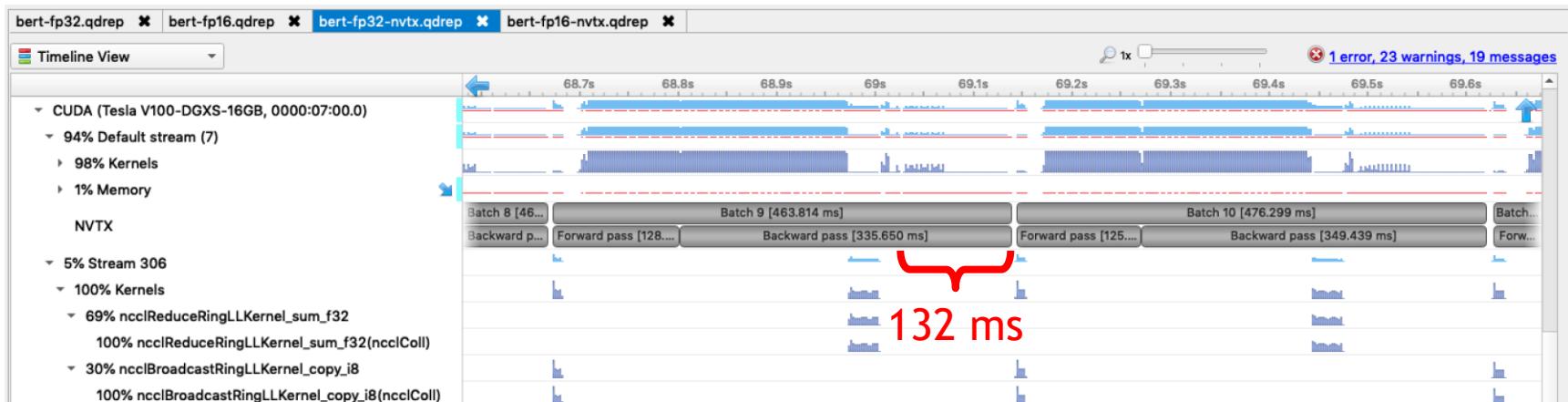


- volta_fp16_s884gemm indicates for using the tensor cores.

ALGORITHM OPTIMIZATION - TRAINING

APEX ADAM Optimizer Optimization

- ADAM optimizer
 - Low Utilization
- APEX ADAM optimizer
 - High utilization



<https://github.com/NVIDIA/apex>



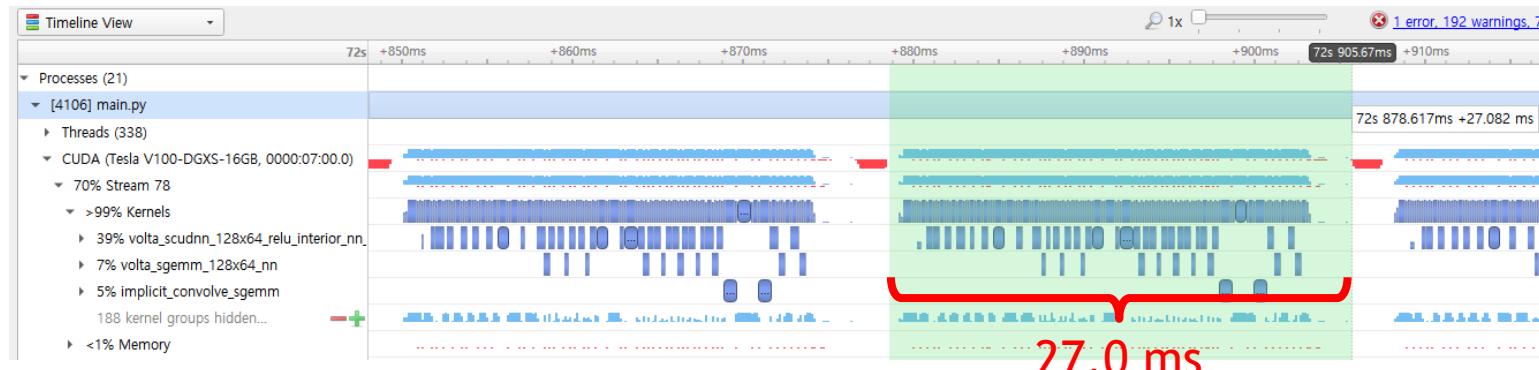
INFERENCE OPTIMIZATION CASE

ALGORITHM OPTIMIZATION - INFERENCE

TensorFlow Single Precision (FP32) vs. half Precision (FP16)

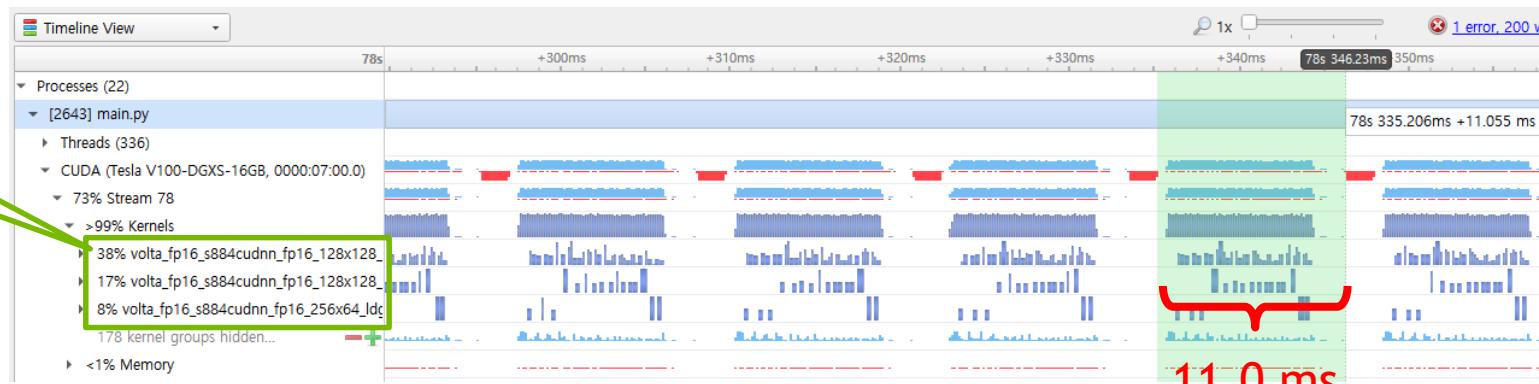
- 2.4x Speed up (~27.0ms vs. ~11.0ms)

Single Precision (FP32)



Tensor Cores
APIs

Half Precision (FP16)

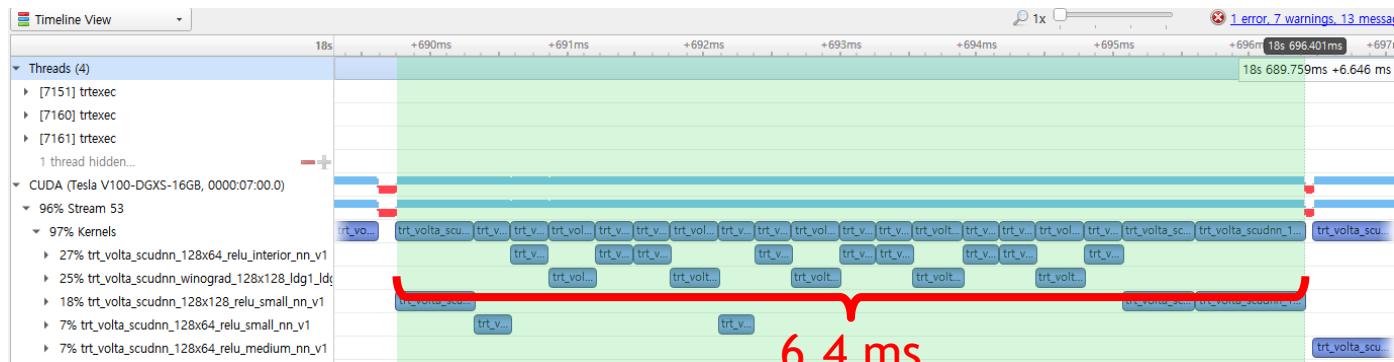


ALGORITHM OPTIMIZATION - INFERENCE

TensorRT Single Precision (FP32) vs. Half Precision (FP16)

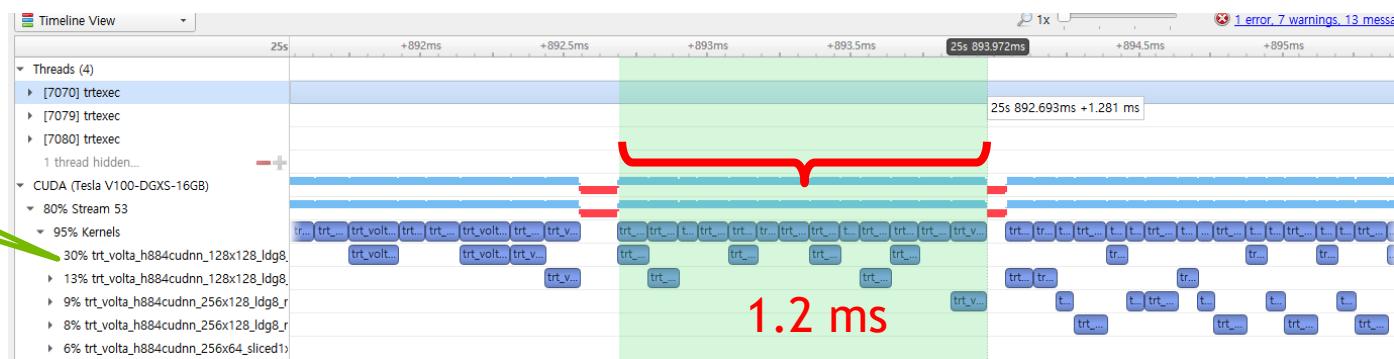
- 5.3x Speed up (~6.4ms vs. ~1.2ms)

Single Precision (FP32)



Tensor Cores
APIs

Half Precision (FP16)





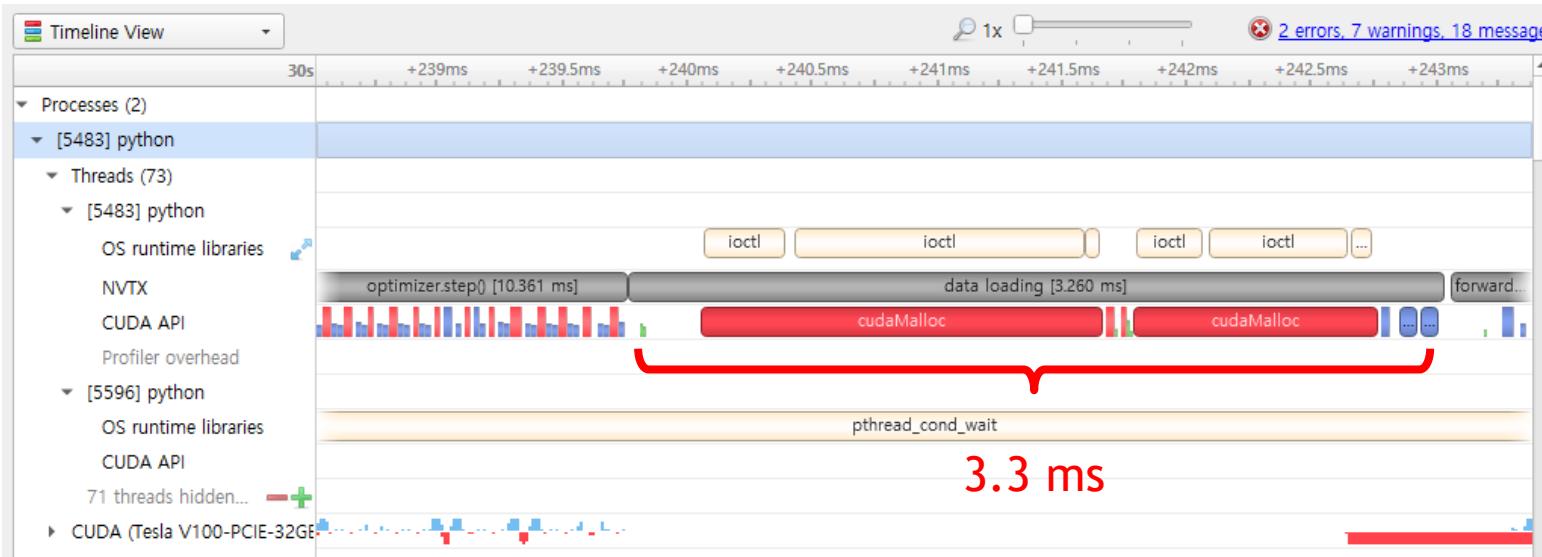
DATA PIPELINE OPTIMIZATION CASE

DATA PIPELINE OPTIMIZATION

Naïve data augmentation pipeline

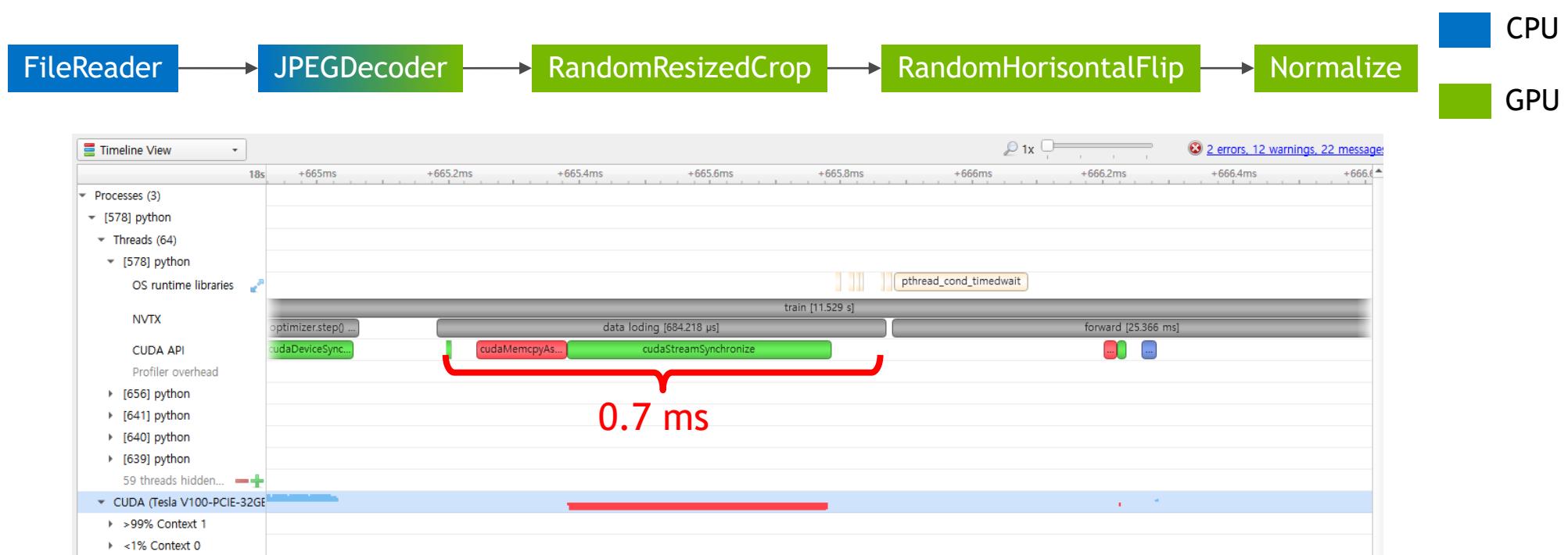


CPU
GPU



DATA PIPELINE OPTIMIZATION

DALI (Data Loading Library)



4.7x Speed up (3.3ms vs. ~0.7ms)

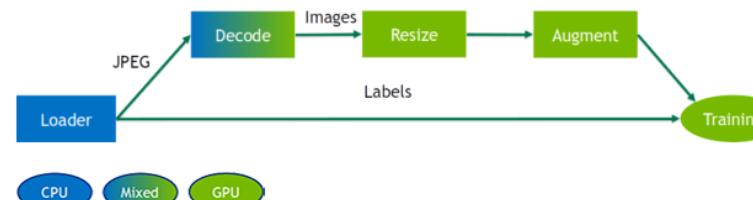
OPTIMIZATION STRATEGY FOR DL

Reminding

- ▶ Algorithm optimization
 - ▶ Tensor Cores
 - ▶ Training: Automatic Mixed Precision
 - ▶ Inference: TensorRT
- ▶ Data pipeline optimization
 - ▶ DALI (Data loading Library)

$$D = \begin{pmatrix} A_{0,0} & A_{0,1} & A_{0,2} & A_{0,3} \\ A_{1,0} & A_{1,1} & A_{1,2} & A_{1,3} \\ A_{2,0} & A_{2,1} & A_{2,2} & A_{2,3} \\ A_{3,0} & A_{3,1} & A_{3,2} & A_{3,3} \end{pmatrix}_{\text{FP16 or FP32}} \begin{pmatrix} B_{0,0} & B_{0,1} & B_{0,2} & B_{0,3} \\ B_{1,0} & B_{1,1} & B_{1,2} & B_{1,3} \\ B_{2,0} & B_{2,1} & B_{2,2} & B_{2,3} \\ B_{3,0} & B_{3,1} & B_{3,2} & B_{3,3} \end{pmatrix}_{\text{FP16}} + \begin{pmatrix} C_{0,0} & C_{0,1} & C_{0,2} & C_{0,3} \\ C_{1,0} & C_{1,1} & C_{1,2} & C_{1,3} \\ C_{2,0} & C_{2,1} & C_{2,2} & C_{2,3} \\ C_{3,0} & C_{3,1} & C_{3,2} & C_{3,3} \end{pmatrix}_{\text{FP16 or FP32}}$$

- ▶ Available in Volta and Turing architecture GPUs
- ▶ 125 Tflops in FP16 vs. 15.7 Tflops in FP32 (8x speed-up)
- ▶ Optimized 4x4x4 dot operation (GEMM)



DALI example pipeline



GETTING STARTED RESOURCES

LEARN MORE

- Nsight System
 - <https://developer.nvidia.com/nsight-systems>
- Official Documentation (Nsight System developer guide)
 - <https://docs.nvidia.com/nsight-systems/>
- Nsight System Blog
 - <https://devblogs.nvidia.com/nsight-systems-exposes-gpu-optimization/>

LEARN MORE DURING AI CONFERENCE

- ▶ 17:20 ~ 18:00 Track2
 - ▶ 효율적인 Deep Learning 서비스 구축을 위한 핵심 애플리케이션 - NVIDIA TensorRT Inference Server by NVIDIA 정소영 상무

RELATED SESSIONS

Automatic Mixed Precision (AMP) for training optimization

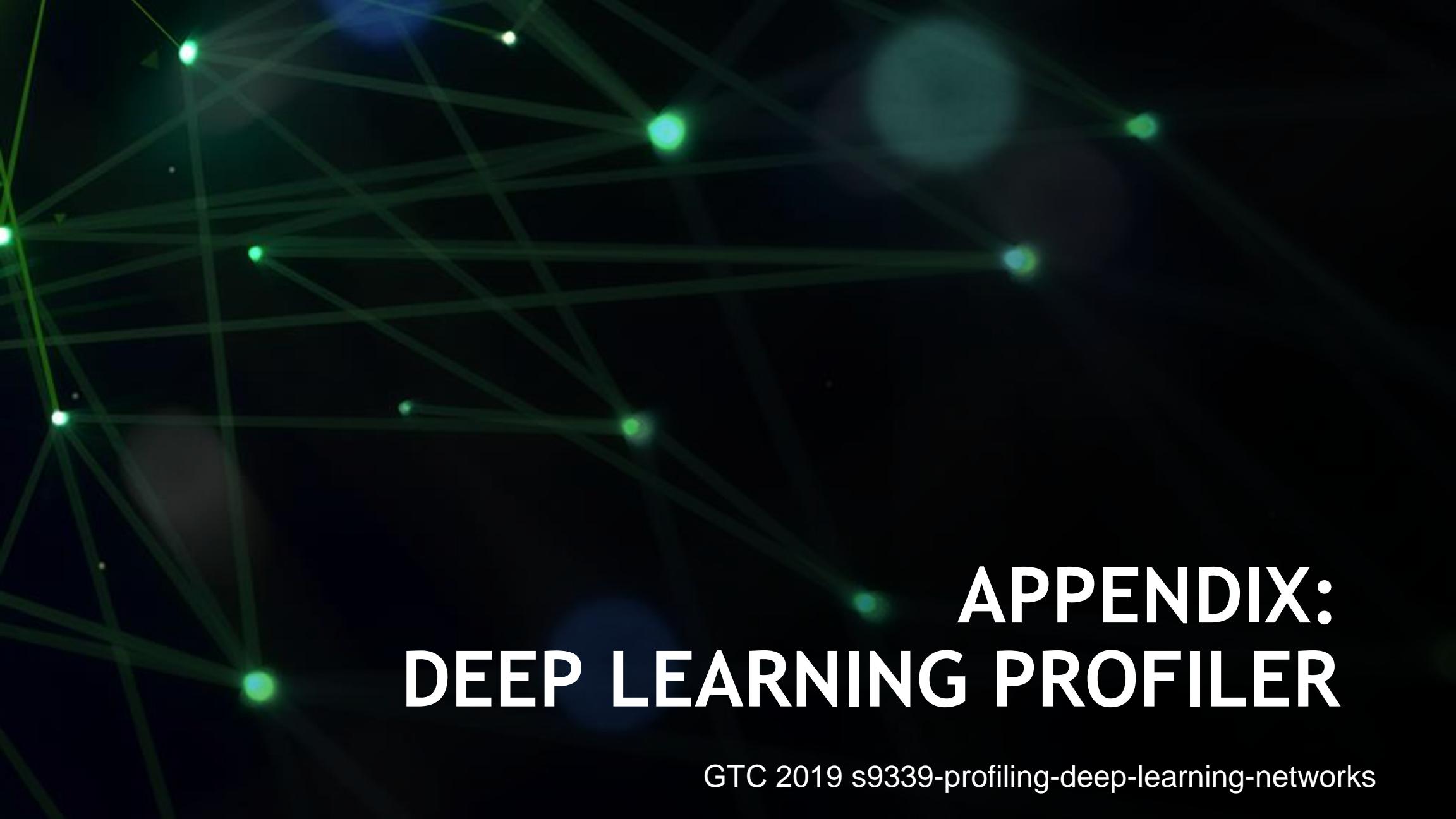
- ▶ 13:00 - 13:40 Track1
 - ▶ Tensor Core를 이용한 딥러닝 학습 가속을 쉽게 하는 방법
(Getting more DL Training Acceleration using Tensor Cores and AMP) by NVIDIA 한재근 과장

TensorRT for inference optimization

- ▶ 13:50 ~ 14:30 Track2
 - ▶ Deep Learning inference 가속화를 위한 NVIDIA의 기술 소개 by NVIDIA 이종환 과장
- ▶ 14:40 ~ 15:20 Track2
 - ▶ TensorRT를 이용한 OCR Model Inference 성능 최적화 by KAKAO 이현수

DALI for data pipeline optimization

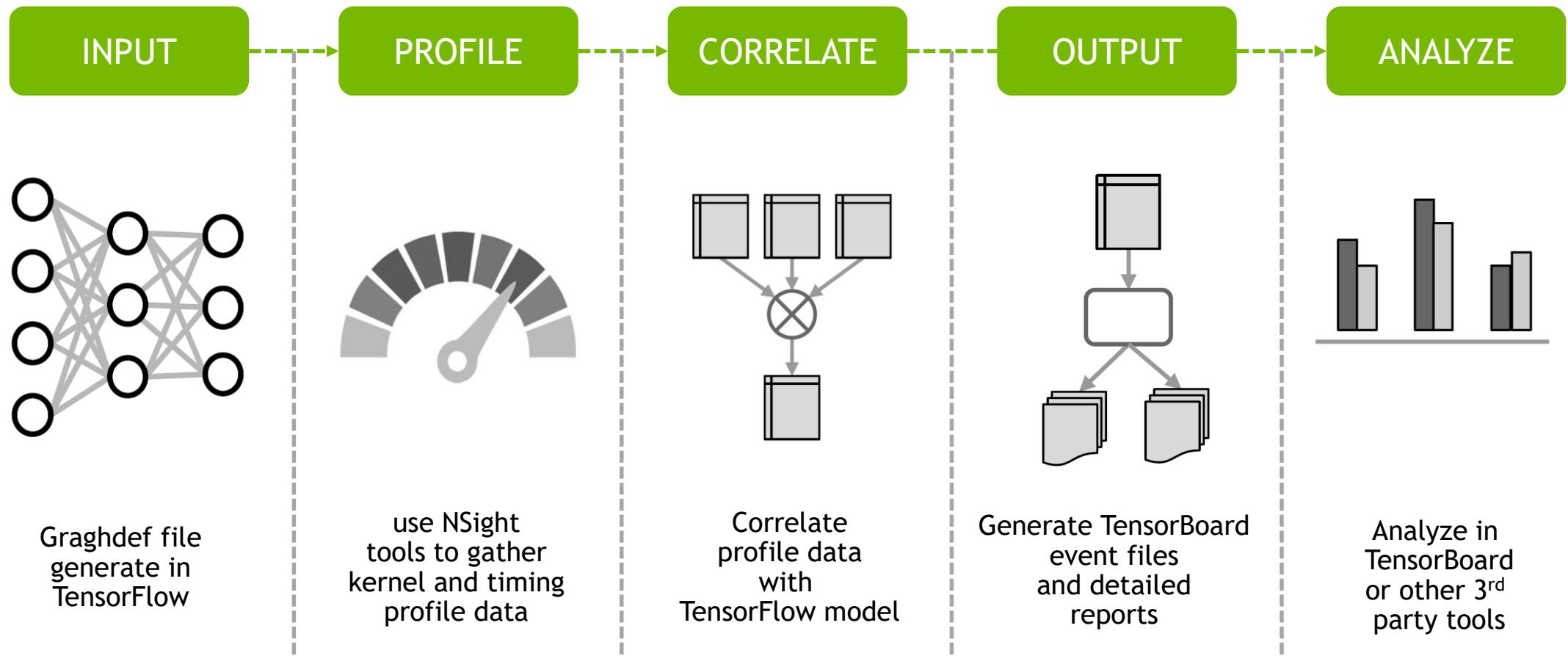
- ▶ 15:40 ~ 16:20 Track1
 - ▶ GPU를 활용한 Image Augmentation 가속화 방안 - DALI by NVIDIA 한재근 과장



APPENDIX: DEEP LEARNING PROFILER

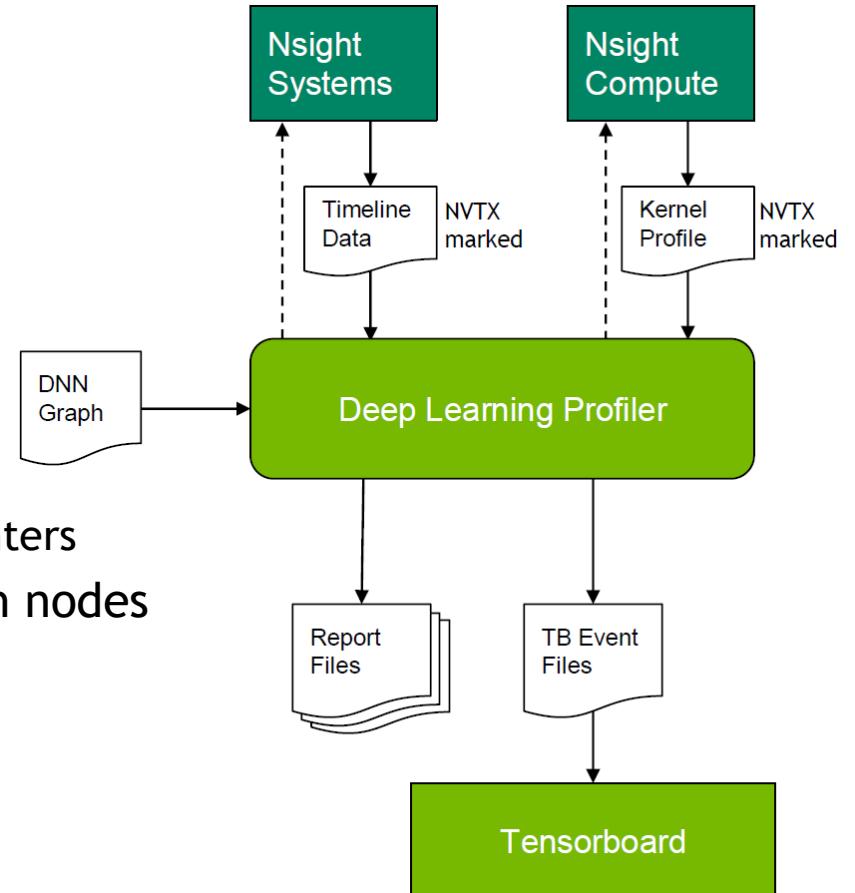
GTC 2019 s9339-profiling-deep-learning-networks

DEEP LEARNING PROFILER WORKFLOW



ARCHITECTURE

- ▶ Automates workflow
- ▶ Nsight Systems
 - ▶ Gather timeline information
 - ▶ Determines Tensor Core usage from name of kernels
- ▶ Nsight Compute
 - ▶ Detailed kernel level profiling
 - ▶ Determines Tensor Core usage from GPU program counters
- ▶ Use NVTX markers to correlate kernels with DNN graph nodes
- ▶ Any number of reports can be generated
 - ▶ TB event Files, CSV, JSON
 - ▶ Analyze with tool of your choice



DEEP LEARNING PROFILER

Command Line Example

- ▶ Example command to profile mobileNet V2 and generate a graphdef

```
$ /usr/bin/python tf_cnn_benchmarks.py --num_gpus=1 --batch_size=8 --model=mobilenet --device=gpu --gpu_indices=1 --data_name=imagenet --data_dir=/data/train-val-tfrecord-480 --num_batches=1 --use_fp16 --fp16_enable_auto_loss_scale --graph_file=/results/mobilenet_graph.pb
```

- ▶ Example Deep Learning Profiler command

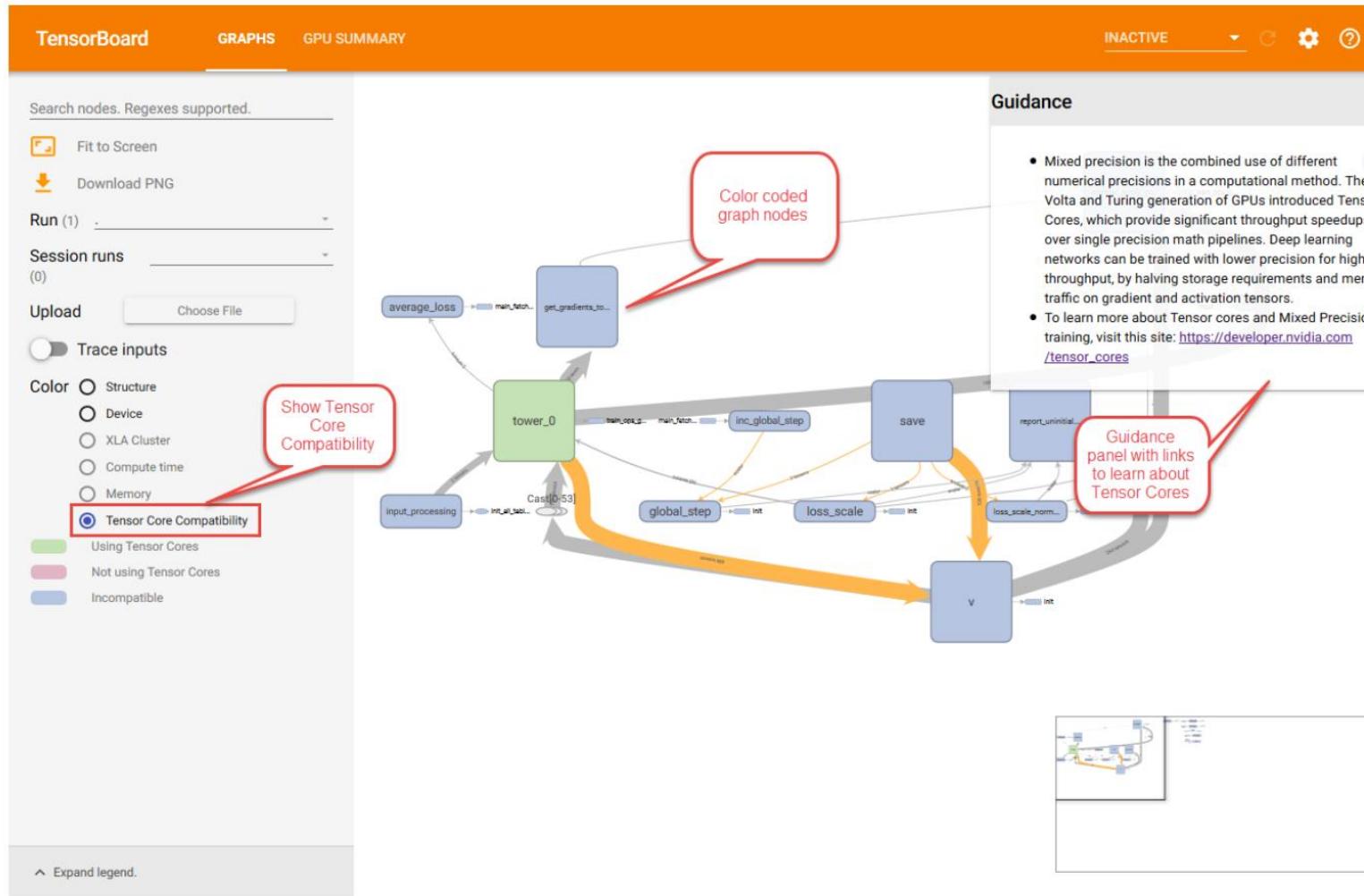
```
$ dlprof --in_graphdef=/results/mobilenet_graph.pb /usr/bin/python tf_cnn_benchmarks.py --num_gpus=1 --batch_size=8 --model=mobilenet --device=gpu --gpu_indices=1 --data_name=imagenet --data_dir=/data/train-val-tfrecord-480 --num_batches=1 --use_fp16 --fp16_enable_auto_loss_scale
```

- ▶ Launching TensorBoard

```
$ tensorboard --logdir ./event_files
```

TENSORBOARD MODIFICATIONS

Start TensorBoard with NVIDIA modifications



COMPATIBILITY DETAILS

Select Compatible using Tensor Cores

TensorBoard GRAPHS GPU SUMMARY INACTIVE

Search nodes. Regexes supported.

Fit to Screen Download PNG

Run (1)

Session runs (0)

Upload Choose File

Trace inputs

Color Structure Device XLA Cluster Compute time Memory **Tensor Core Compatibility**

- Using Tensor Cores
- Not using Tensor Cores
- Incompatible

Expand legend.

The interface shows a graph of operations (BatchNorm, DepthwiseConv2D, etc.) and their dependencies. A node in the graph is highlighted with a red circle. A callout bubble points to it with the text "Click on a node that uses Tensor Cores". Another callout bubble points to the "Compatibility Details" section with the text "See list of all kernels called". A third callout bubble points to the table with the text "Statics on the node runtime".

Compatibility Details

Duration: 4.78 (ms)
Kernels on this node: 17

Name	Using TC	Duration (μs)	Calls	Avg (μs)	Min (μs)	Max (μs)
volta_fp16_ncnhw_nn_v1	YES	97	13	7	5	9
compute	volta_fp16_s884cudnn_fp16_128x128_Idg8_relu_f2f_exp_intero	100	15	7	5	8
void cudnn..._half*)	no	53	4	13	9	18
void cudnn..._int, int)	no	51	4	13	10	16
void fft1d..._half()	no	18	4	4	4	5
void fft1d..._nt2, int2)	no	19	4	5	5	5
void fft1d..._nt2, int2)	no	30	4	8	7	8
void fft2d..._half()	no	19	4	5	5	5
void fft2d..._int, int)	no	37	8	5	4	5
void flip..._int, int)	no	31	4	8	7	9
void im2co...alf*, int)	no	29	4	7	7	8
void nchwT...at, float)	no	132	13	10	8	11
void tenso...en::half*)	no	88	11	8	7	9
volta_cgemm_32x64_tn	no	21	4	5	5	6
volta_fp16_ror_nn_v1	no	11	2	5	5	6
volta_cgemm_64x32_nt	no	21	4	5	5	6

COMPATIBILITY DETAILS

Select Compatible using Tensor Cores

Compatibility details and panel providing guidance and links to help with mixed precision

TensorBoard GRAPHS GPU SUMMARY INACTIVE

Search nodes. Regexes supported.

Fit to Screen Download PNG

Run (1) Session runs (0)

Upload Choose File

Trace inputs

Color: Structure, Device, XLA Cluster, Compute time, Memory, **Tensor Core Compatibility** (Using Tensor Cores, Not using Tensor Cores, Incompatible)

Expand legend.

See a Tensor Core Help panel with useful link for more information

Remove from main graph

Compatibility Details

Duration: 521.88 (μs)
Kernels on this node: 2

Name	Using TC	Duration (μs)	Calls	Avg (μs)	Min (μs)	Max (μs)
void cuDNN... int, int)	no	165	11	15	13	16
void tensor...enhalf*)	no	85	11	8	7	8

Tensor Core Help

- Please note that if there are multiple kernels being observed on single op node, these are likely performing data transposes to prepare the data for efficient use by tensorcores. Such transposes themselves would not use tensorcores.
- To learn more about Tensor cores and Mixed Precision training, visit this site: https://developer.nvidia.com/tensor_cores.
- You will find resources on how to train networks with mixed precision and make full use of Tensor cores for Tensorflow models https://docs.nvidia.com/deeplearning/sdk/mixed-precision-training/index.html#training_tensorflow.
- Visit this site to find out more about DNN examples optimized and tuned for Tensor cores provided by NVIDIA: <https://developer.nvidia.com/deep-learning-examples>.

OPNODES SUMMARY TAB

GPU Summary tab showing all the Nodes, compatible and using Tensor Cores

The screenshot shows the TensorBoard interface with the 'GPU SUMMARY' tab selected. A red box highlights the 'New GPU Summary Tab' button in the top-left corner. Another red box highlights the 'See a sortable list of all nodes in the model' text next to the 'Run' dropdown. A third red box highlights the 'See a list of all kernels for a selected node' text next to the 'Kernel Name' column in the bottom table. The main area displays two tables: one listing nodes and another listing kernels for a selected node.

● OpNodes ● GroupNodes ● Model Summary

INACTIVE

New GPU Summary Tab

Search

Run (1)

Session runs (0)

Upload Choose File

See a sortable list of all nodes in the model

See a list of all kernels for a selected node

Counter	Kernel Name	Using TC	Duration (µs)	Calls	Average (µs)	Minimum (µs)	Maximum (µs)
1	volta_s884cudnn_fp16_128x128_ldg8_relu_exp_interior_nhwc_tn_v1	✓	14	2	7	7	7
2	cudnn::gemm::computeOffsetsKernel(cudnn::gemm::ComputeOffsetsParams)	X	122	15	8	4	10
3	void cudnn::detail::explicit_convolve_sgemm<__half, int, 512, 6, 8, 3, 3, 5, 0, true>(int, int, __half const*, int, __half const*, int, __half*, kernel_conv_params, int, int, float, float, int, __half*, __half*)	X	51	4	13	9	17
4	void cudnn::detail::implicit_convolve_sgemm<__half, __half, 512, 6, 8, 3, 3, 5, 1, true, false, true>(int, int, int, __half const*, int, __half*, __half*, kernel_conv_params, int, float, float, int, __half*, __half*, int, int)	X	54	4	14	10	18

GROUP NODE SUMMARY TAB

Roll up timing metrics and Tensor Core utilization per group node

The screenshot shows the TensorBoard interface with the GPU SUMMARY tab selected. On the left, there's a sidebar with options like 'Search nodes. Regexes sup...', 'Fit to screen', 'Download PNG', 'Run (1)', 'Session runs (0)', and an 'Upload' section. A red box highlights the 'List Group Nodes in model' button. The main area has tabs for 'OpNodes' (radioed), 'GroupNodes' (selected), and 'Model Summary'. The 'GroupNodes' tab displays a table of nodes and their metrics. A red box highlights the 'Sort by total time' button. Another red box highlights the text 'Use statistics to drill down into bottlenecks'.

Node Path	Total Time	Tensor Cores	Ops	Memory
1 input_processing/batch_processing	6,179,353	0	0	0
2 input_processing	6,179,353	0	0	0
3 input_processing/batch_processing/list_files	6,140,401	0	0	0
4 tower_0/v	2,585,470	111	97	87.387
5 tower_0	2,585,470	111	97	87.387
6 tower_0/v/cg	2,097,763	37	31	83.784
7 tower_0/v/cg/MobilenetV2	2,097,182	36	31	86.111
8 tower_0/v/cg/MobilenetV2/Conv	1,850,516	1	1	100
9 tower_0/v/gradients	420,913	74	66	89.189
10 tower_0/v/gradients/tower_0/v	394,305	74	66	89.189
11 tower_0/v/gradients/tower_0	394,305	74	66	89.189
12 tower_0/v/gradients/tower_0/v/cg	371,357	74	66	89.189
13 tower_0/v/gradients/tower_0/v/cg/MobilenetV2	369,520	72	66	91.667
14 tower_0/v/gradients/tower_0/v/cg/MobilenetV2/Logits	63,684	2	2	100
15 tower_0/v/gradients/tower_0/v/cg/MobilenetV2/Logits/Conv2d_1c_1x1	2,865	2	2	100
16 tower_0/v/gradients/tower_0/v/cg/MobilenetV2/Logits/Conv2d_1c_1x1/Conv2D_grad	2,425	2	2	100
17 tower_0/v/cg/MobilenetV2/Logits	3,134	1	1	100
18 tower_0/v/cg/MobilenetV2/Logits/Conv2d_1c_1x1	1,063	1	1	100
19 get_gradients_to_apply	45,687	0	0	0
20 tower_0/v/l2_loss	44,596	0	0	0
21 tower_0/v/gradients/tower_0/v/cg/MobilenetV2/expanded_conv_16	36,684	4	4	100
22 tower_0/v/gradients/tower_0/v/cg/MobilenetV2/expanded_conv_12	32,883	4	4	100
23 tower_0/v/gradients/tower_0/v/cg/MobilenetV2/Conv_1	27,752	2	2	100
24 tower_0/v/gradients/tower_0/v/cg/MobilenetV2/Conv_1/Conv2D_grad	27,046	2	2	100
25 tower_0/v/gradients/tower_0/v/cg/MobilenetV2/expanded_conv_11	24,602	4	4	100
26 v/cg	24,131	0	0	0

MODEL SUMMARY TABLE

Model Summary shows concise information on Tensor core usage

TensorBoard GRAPHS GPU SUMMARY

Search nodes. Regexes sup...

 Fit to screen
 Download PNG

Run
(1)

Session runs (0)

Upload

OpNodes GroupNodes Model Summary

Nodes in Graph	6158
Compatible Nodes	112
Compatible Nodes using Tensor Cores	97
Total GPU Time	15.10 (s)
% of time in TC compatible nodes	15.62%



NVIDIA®

