PROFILING DEEP LEARNING NETWORK

Jack Han | Solutions Architect | jahan@nvidia.com

AGENDA

Profiling Deep Learning Network

- Introduction to Nsight Systems

PyTorch NVTX Annotation

- Examples and Benefits
- Mixed Precision with BERT

TensorFlow NVTX Plugins

- Plugin Usage Example

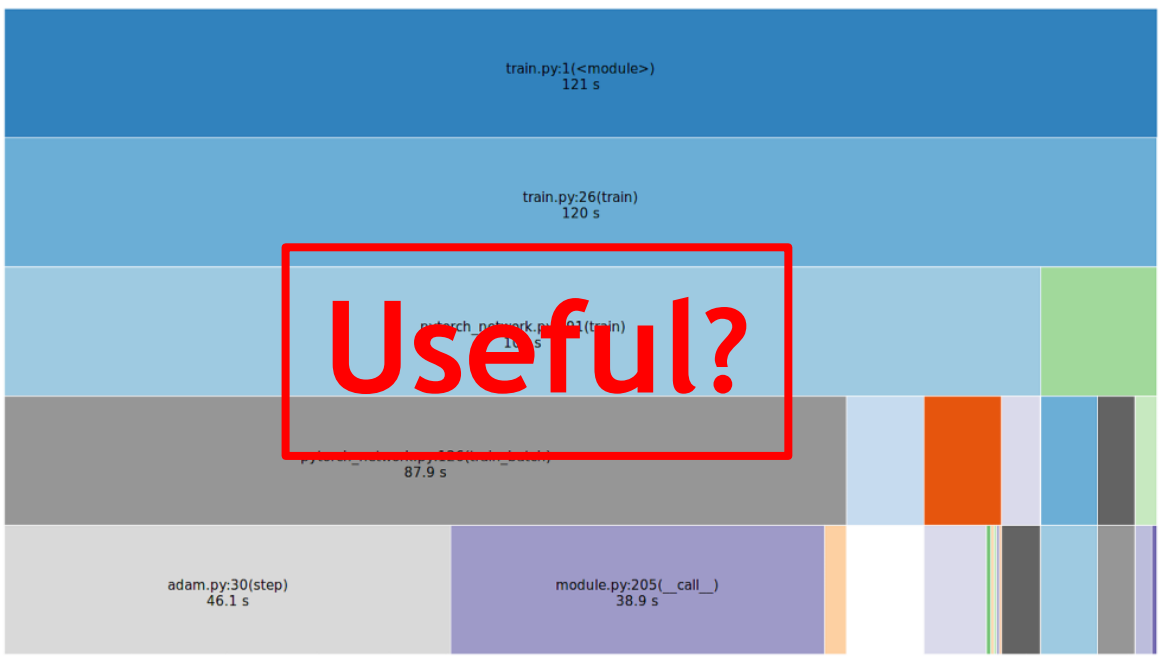
HOW TO SPEED-UP NETWORK

- 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



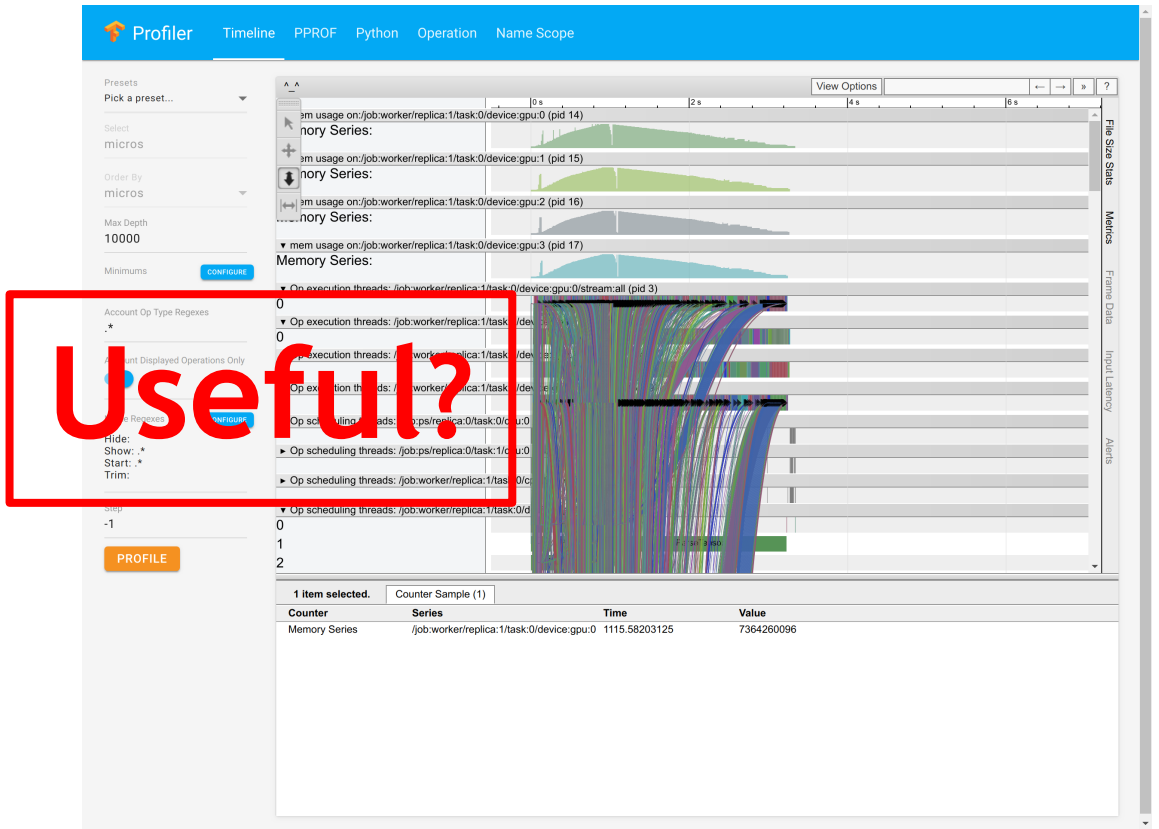

```
python -m cProfile -o 100_percent_gpu_utilization.prof train.py
snakeviz 100_percent_gpu_utilization.prof
```

https://sagivtech.com/2017/09/19/optimizing-pytorch-training-code/

TENSORFLOW PROFILER

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

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



NVIDIA PROFILING TOOLS

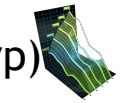
Nsight Systems 

Nsight Compute 

Nsight Visual Studio Edition

Nsight Eclipse Edition 

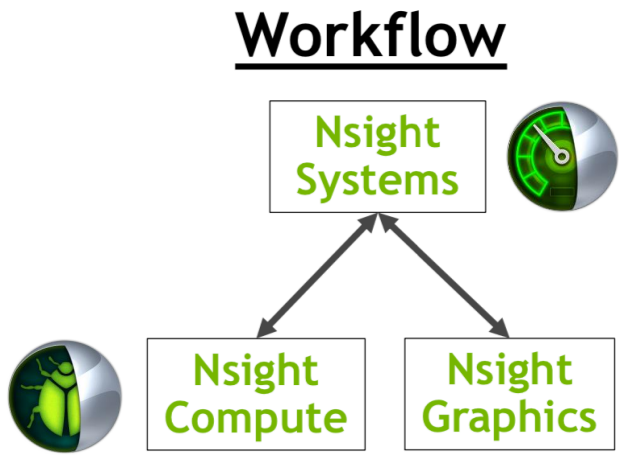
NVIDIA Profiler (nvprof)

NVIDIA Visual Profiler (nvvp) 

CUPTI (CUDA Profiling Tools Interface)

NSIGHT PRODUCT FAMILY

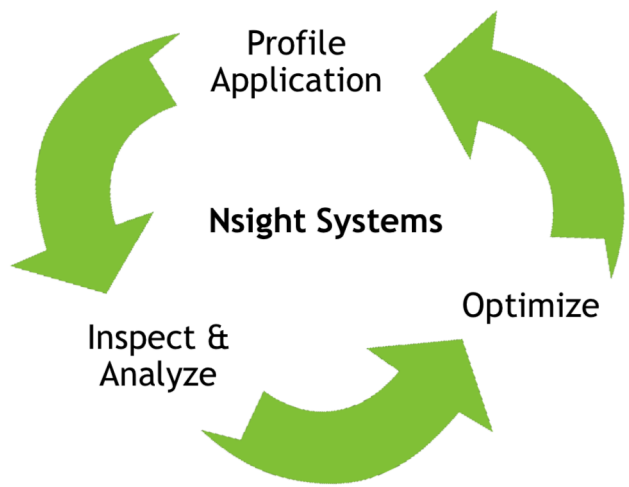
- Standalone Performance Tools
  - Nsight Systems system-wide application algorithm tuning
  - Nsight Compute Debug/optimize specific CUDA kernel
  - Nsight Graphics Debug/optimize specific graphics
- IDE plugins
  - Nsight Eclipse Edicion/Visual Studio editor, debugger, some perf analysis

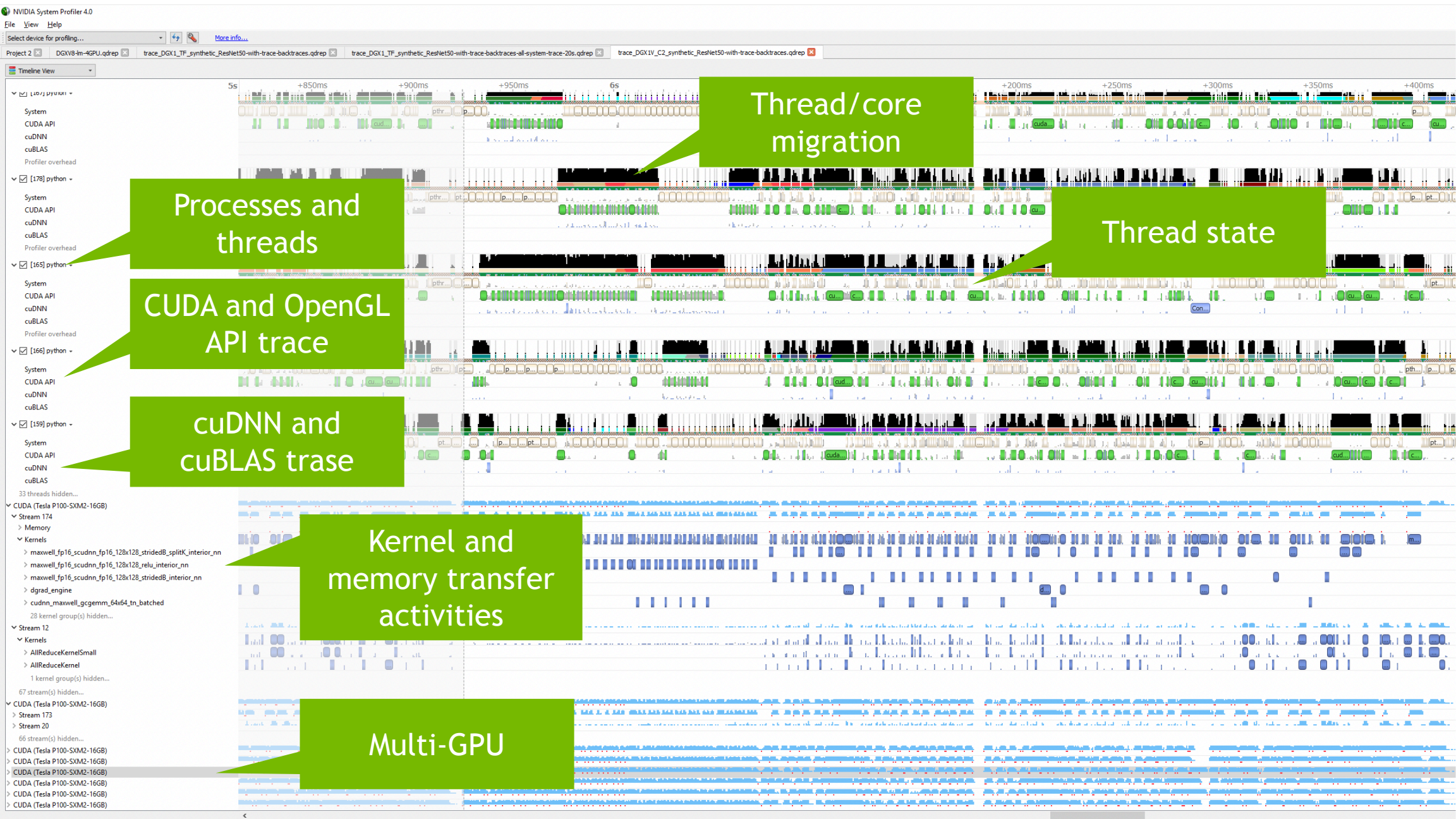


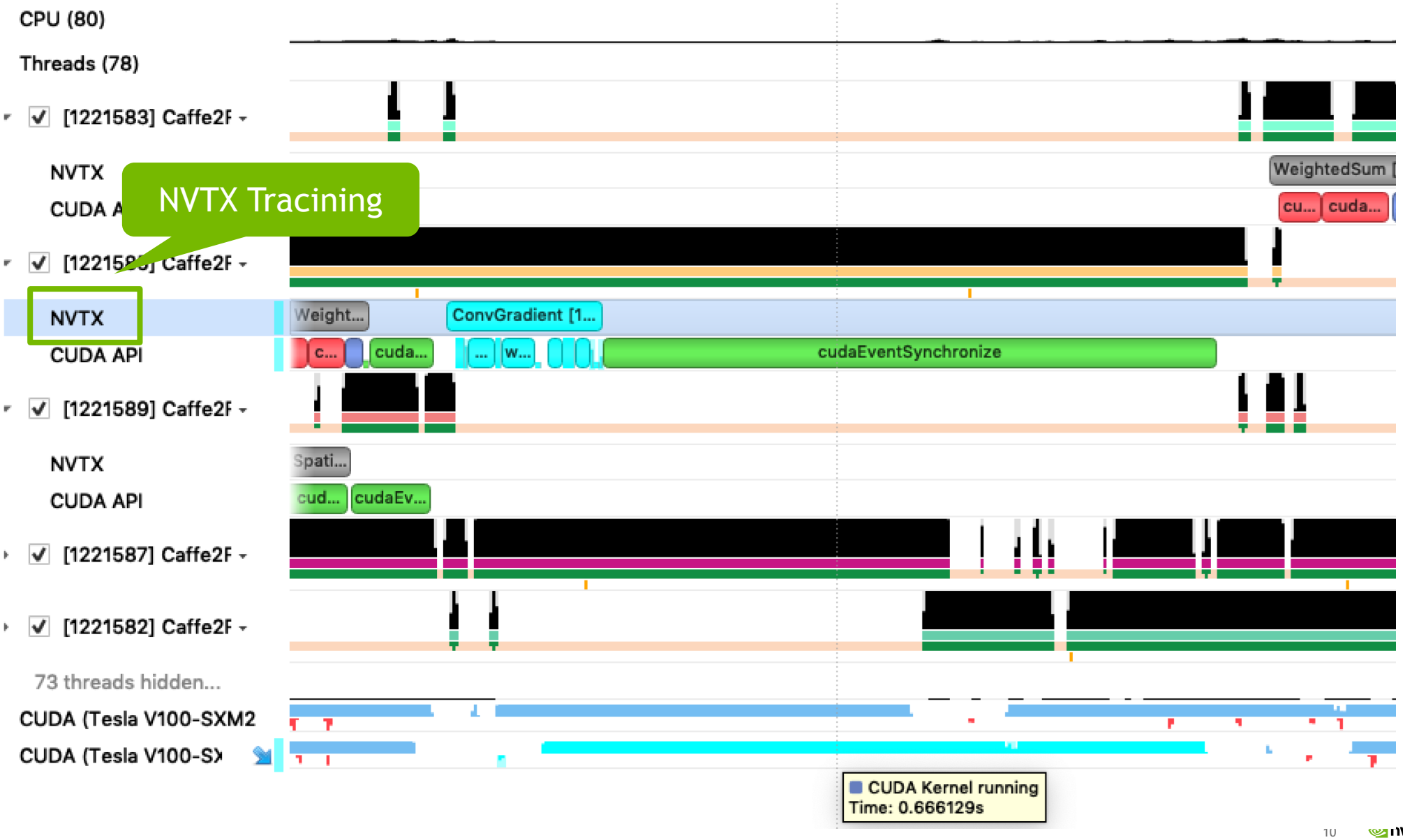
NSIGHT SYSTEMS

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

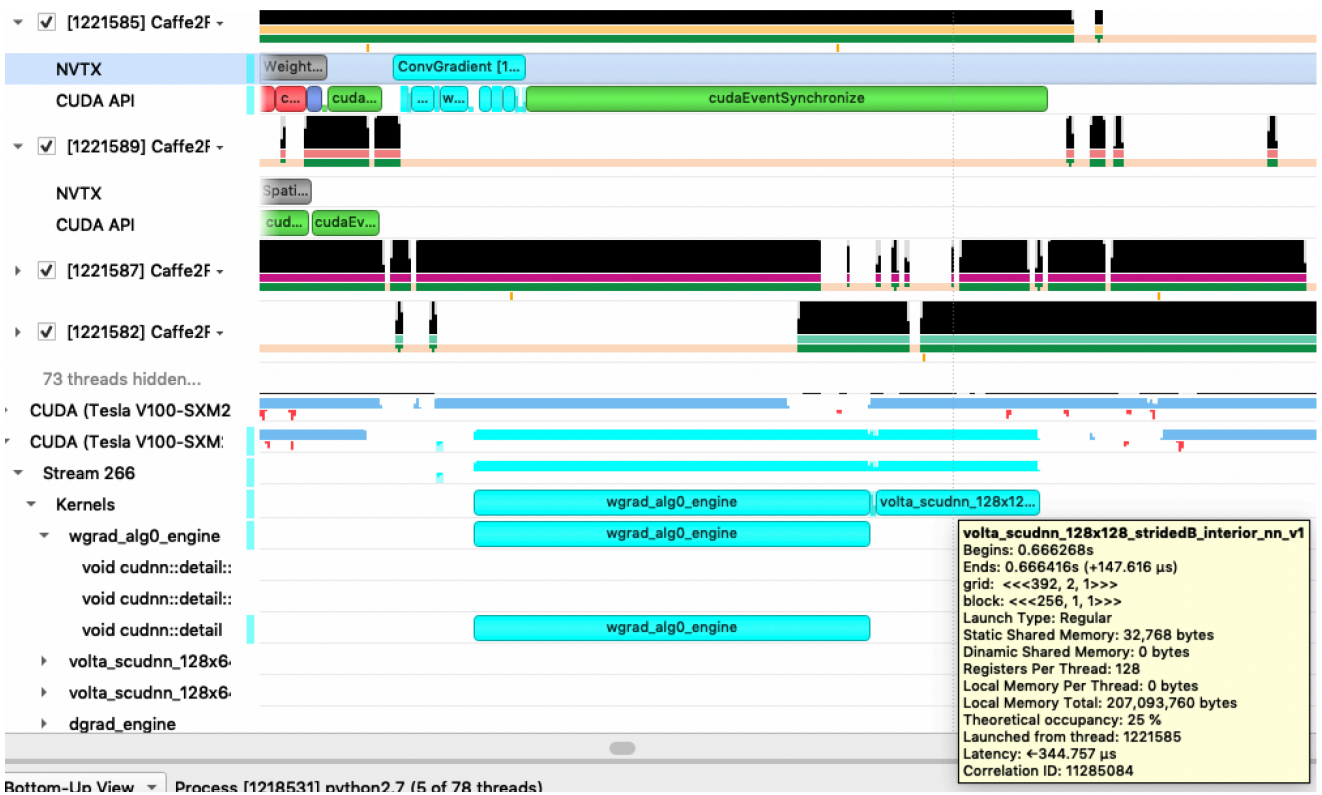



NVIDIA System Profiler 4.0
File View Help
Select device for profiling...
More info...
Project 2
DGX1-lm-4GPU.qdrep
trace_DGX1_TF_synthetic_ResNet50-with-trace-backtraces.qdrep
trace_DGX1_TF_synthetic_ResNet50-with-trace-backtraces-all-system-trace-20s.qdrep
trace_DGX1V_C2_synthetic_ResNet50-with-trace-backtraces.qdrep
Timeline View
5s
+850ms
+900ms
+950ms
6s
+200ms
+250ms
+300ms
+350ms
+400ms
System
CUDA API
cuDNN
cuBLAS
Profiler overhead
[178] python
[165] python
[166] python
[159] python
33 threads hidden...
CUDA (Tesla P100-SXM2-16GB)
Stream 174
Memory
Kernels
maxwell_fp16_scudnn_fp16_128x128_stridedB_splitK_interior_nn
maxwell_fp16_scudnn_fp16_128x128_relu_interior_nn
maxwell_fp16_scudnn_fp16_128x128_stridedB_interior_nn
dgrad_engine
cudnn_maxwell_gcgemm_64x64_tn_batched
28 kernel group(s) hidden...
Stream 12
AllReduceKernelSmall
AllReduceKernel
1 kernel group(s) hidden...
67 stream(s) hidden...
Stream 173
Stream 20
66 stream(s) hidden...
CUDA (Tesla P100-SXM2-16GB)
Thread/core migration
Processes and threads
Thread state
CUDA and OpenGL API trace
cuDNN and cuBLAS trase
Kernel and memory transfer activities
Multi-GPU


CPU (80)
Threads (78)
[1221583] Caffe2F
NVTX
CUDA A
NVTX Tracining
WeightedSum
cu...
cuda...
[1221585] Caffe2F
NVTX
CUDA API
Weight...
ConvGradient [1...
c...
cuda...
...
w...
cudaEventSynchronize
[1221589] Caffe2F
NVTX
CUDA API
Spati...
cud...
cudaEv...
[1221587] Caffe2F
[1221582] Caffe2F
73 threads hidden...
CUDA (Tesla V100-SXM2
CUDA (Tesla V100-SX
CUDA Kernel running
Time: 0.666129s

TRANSITIONING TO PROFILE A KERNEL

Dive into kernel analysis



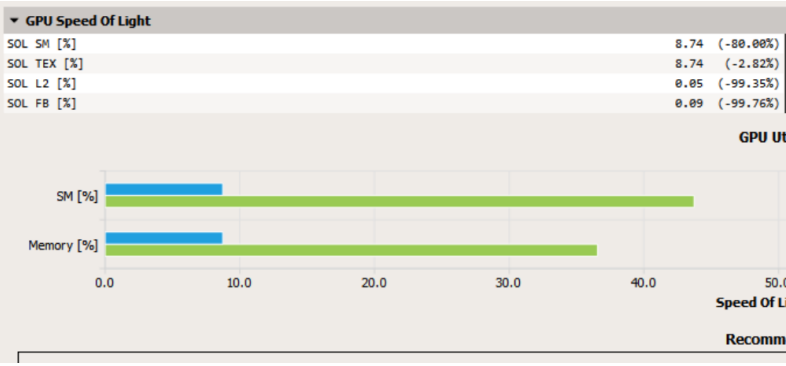
NVIDIA

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_warps [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
launch__waves_per_multiprocessor	3.23	42.11
ltc__sol_pct [%]	6.93	7.18
memory_access_size_type [bytes]	2.00; 32.00; 32.00; 32…	2.00; 32.00; 32.00; 32…

Source Correlation

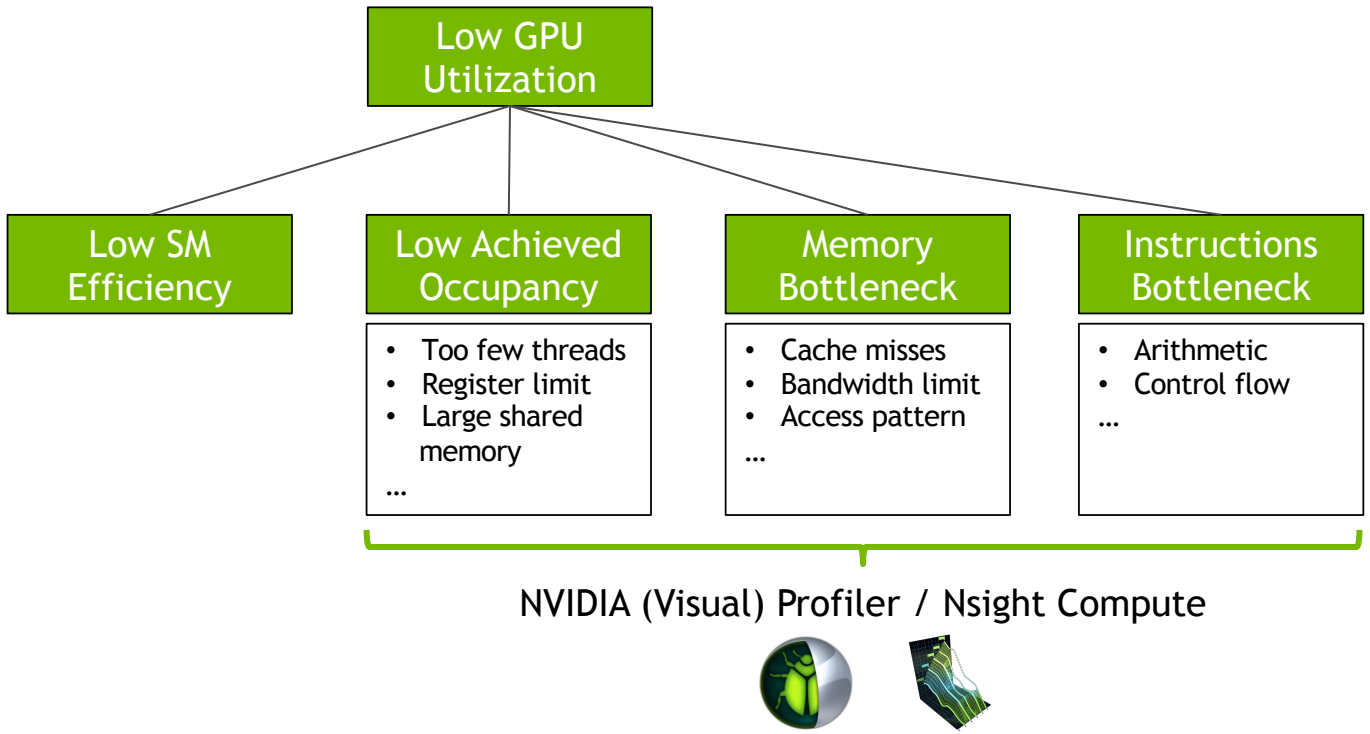
Source	Live Registers	Sampling Data (All)	Sampling Data (No Issue)
@!PT SHFL.IDX PT, RZ, RZ, RZ, RZ;	0	223	0
MOV R1, c[0x0][0x28];	1	13	44
S2R R0, SR_CTAID.X;	2	143	75
S2R R2, SR_TID.X;	3	0	38
IMAD R0, R0, c[0x0][0x0], R2;	3	599	94
ISETP.GE.AND P0, PT, R0, c[0x0][0x170]	2	125	26
@P0 EXIT;	2	259	86
MOV R2, R0;	3	386	29
@!PT SHFL.IDX PT, RZ, RZ, RZ, RZ;	2	0	0
MOV R4, 0x4;	3	0	0
IMAD.WIDE R4, R2, R4, c[0x0][0x160];	4	0	0
LDG.E.SYS R3, [R4];	3	0	0
BSSY B0, 0xb00976780;	3	0	0
SHF.R.S32.HI R0, RZ, 0x1f, R2;	4	0	0

PROFILING GPU APPLICATION

Focusing GPU Computing

How to measure

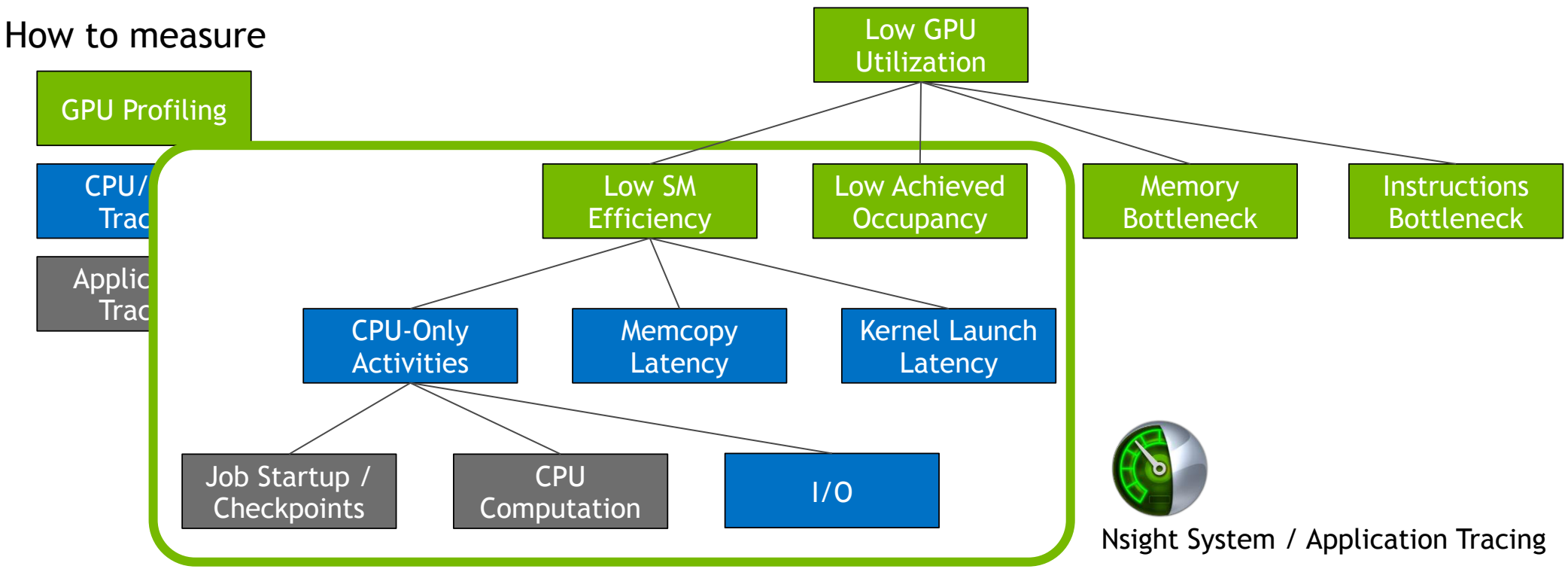
- GPU Profiling
- CPU/GPU Tracing
- Application Tracing



NVIDIA Supports them with cuDNN, cuBLAS, and so on

PROFILING GPU APPLICATION

Focusing System Operation



NSIGHT SYSTEMS PROFILE

Profile with CLI

```
$ nsys profile –t cuda,osrt,nvtx,cudnn,cublas \   ← APIs to be traced
               -y 60 \                            ← Delayed profile (sec)
               -d 20 \                            ← Profiling duration (sec)
               –o baseline \                      ← Output filename
               -f true \                          ← Overwrite when it’s true
               –w true \                          ← Display
               python main.py                     ← Execution command
```

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

DISABLING ASYNCHRONOUS OPERATION

Elimination of CPU interop effects

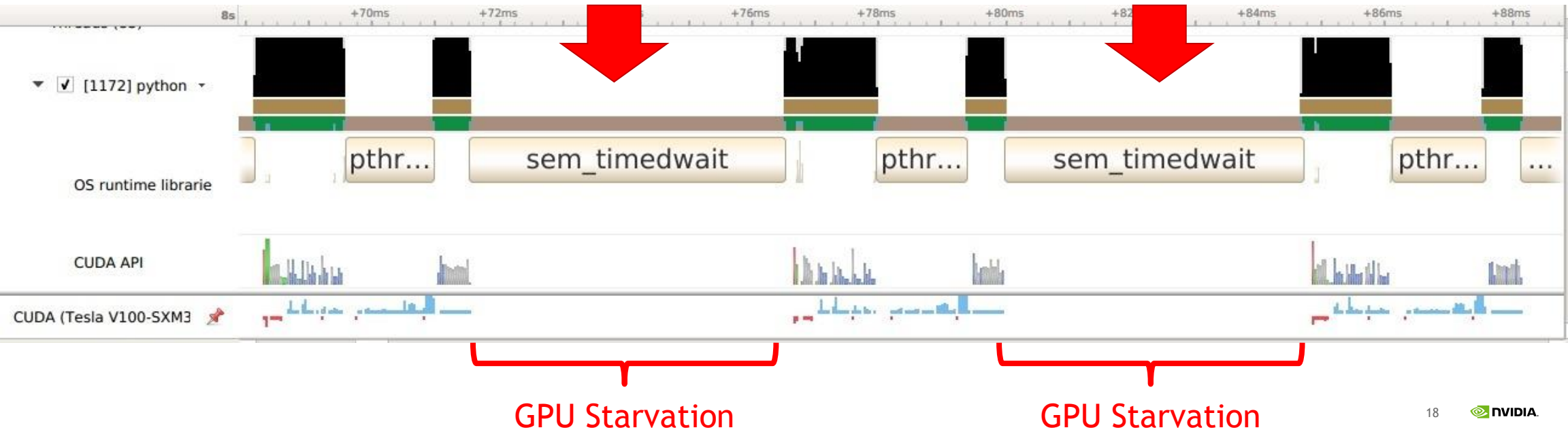
```
export CUDA_LAUNCH_BLOCKING=1
```

https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#asynchronous-concurrent-execution


PROFILING
PYTORCH MODEL

BASELINE PROFILE

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



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)
```

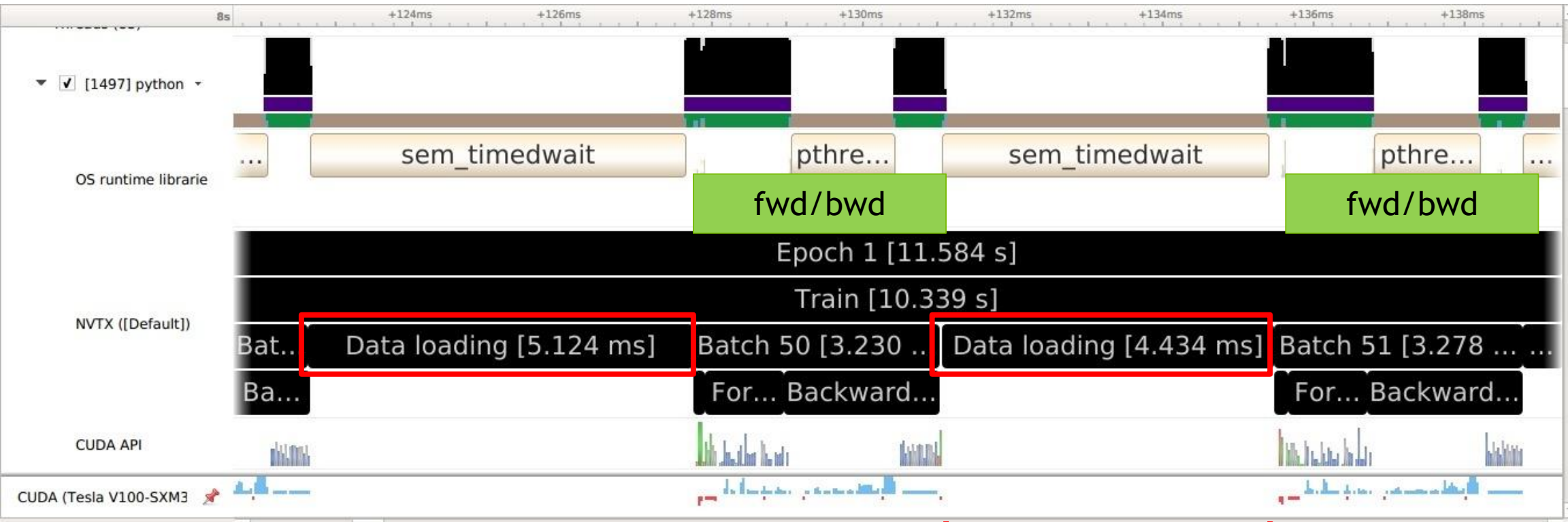
https://pytorch.org/docs/stable/_modules/torch/cuda/nvtx.html

NVTX ANNOTATIONS

```
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()
```

https://pytorch.org/docs/stable/_modules/torch/cuda/nvtx.html

BASELINE PROFILE (WITH NVTX)



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

OPTIMIZE SOURCE CODE

- Data loader was configured to use 1 worker thread

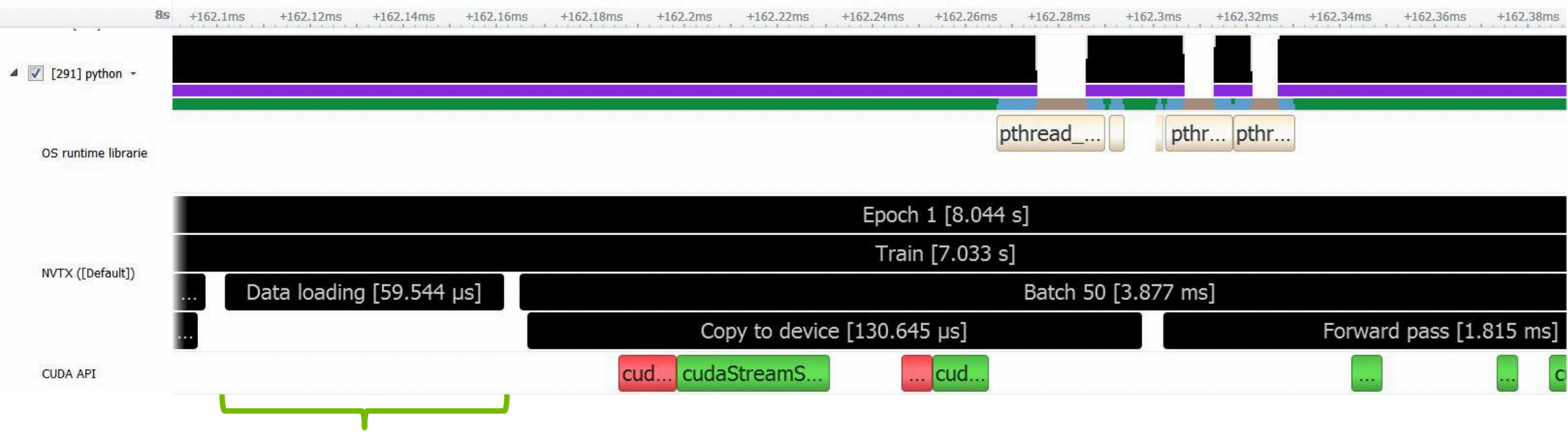
```
kwargs = {'num_workers': 1, 'pin_memory'  True  if use_cuda else {}
```

- Let’s switch to using 8 worker threads:

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

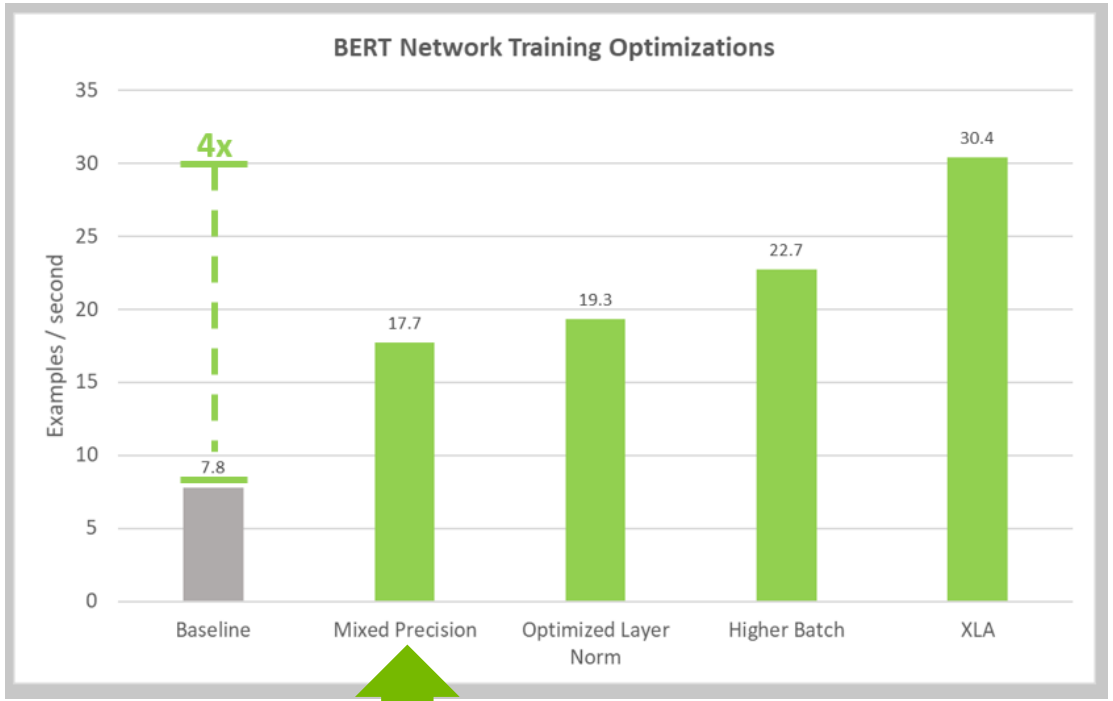
AFTER OPTIMIZATION

- Time for data loading reduced for each bath



Reduced from 5.1ms to 60us for batch 50

REAL EXAMPLE

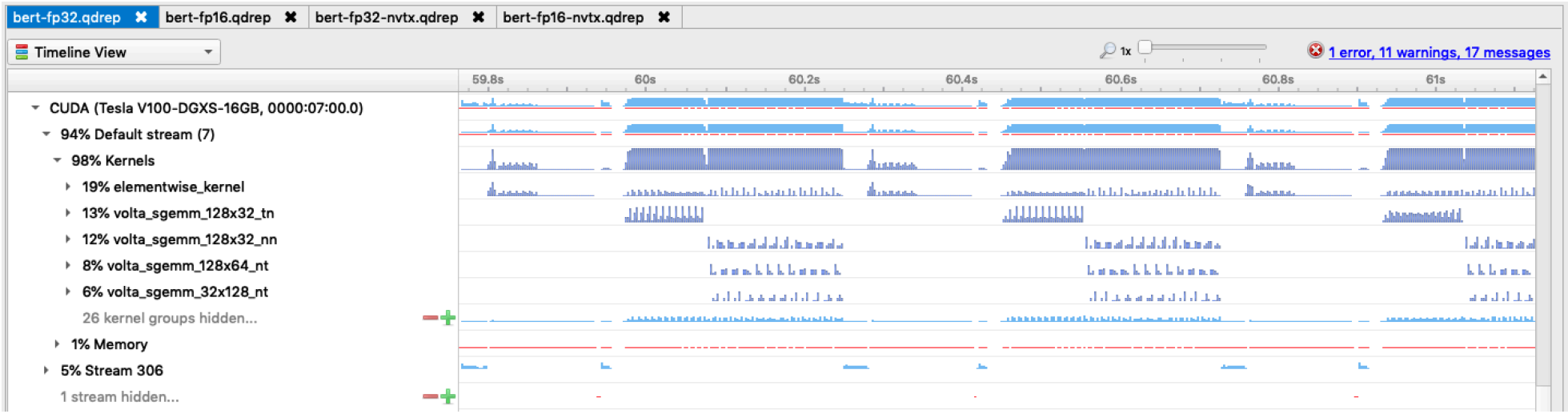


How?

https://news.developer.nvidia.com/nvidia-achieves-4x-speedup-on-bert-neural-network/

INITIAL PROFILE

No NVTX



- Difficult to understand → no useful

NVTX TAGGING

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


...


                                                                                          Backward
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

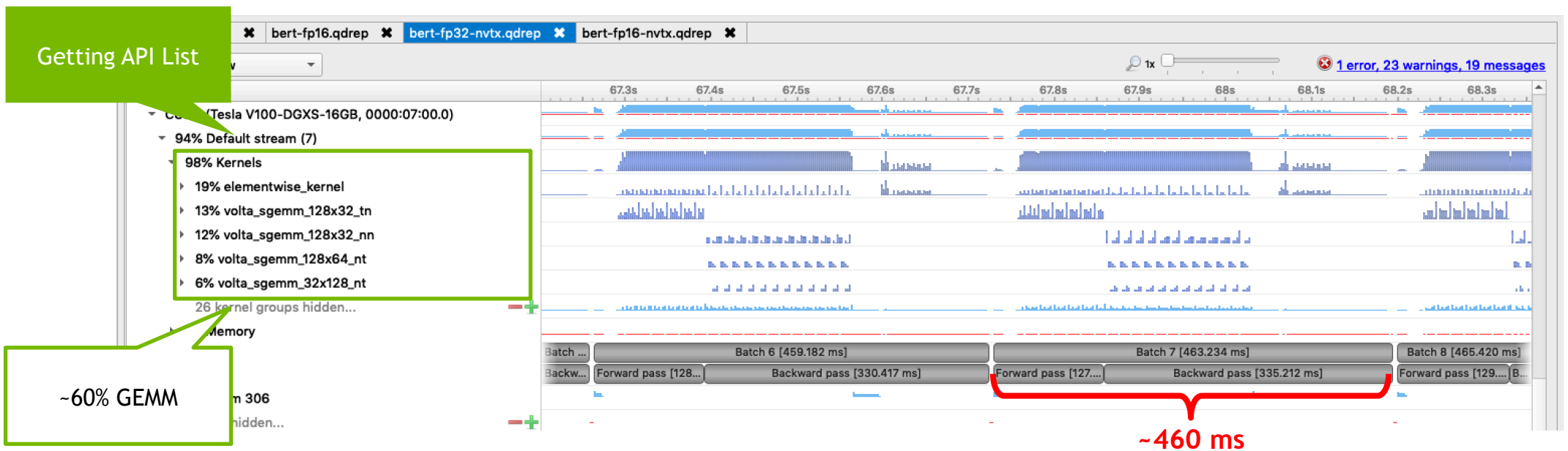
```
nvtx.range_push("Batch " + str(step))
nvtx.range_push("Forward pass")
loss = model(input_ids, segment_ids, input_mask, start_positions, end_positions)
nvtx.range_pop()

...

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()
```

Foward

Backward

BERT FP32 BENCHMARK

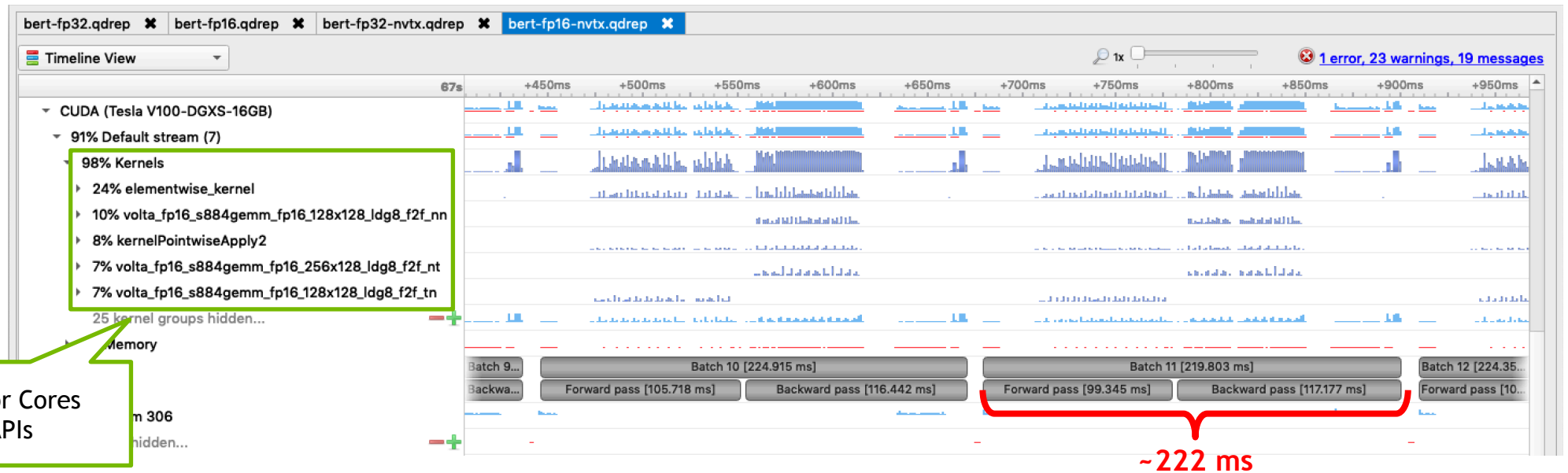
HuggingFace's pretrained BERT



4 V100 GPUs w/ NVLINK, Batch size: 32, max_seq_length: 512

BERT FP16 BENCHMARK

HuggingFace's pretrained BERT

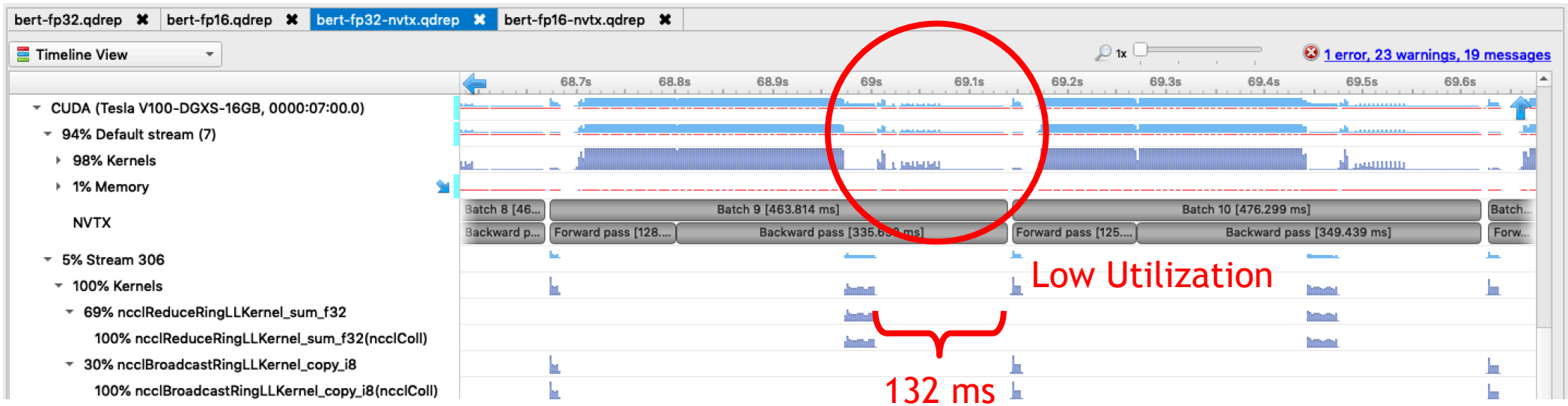


2.1x Speed up

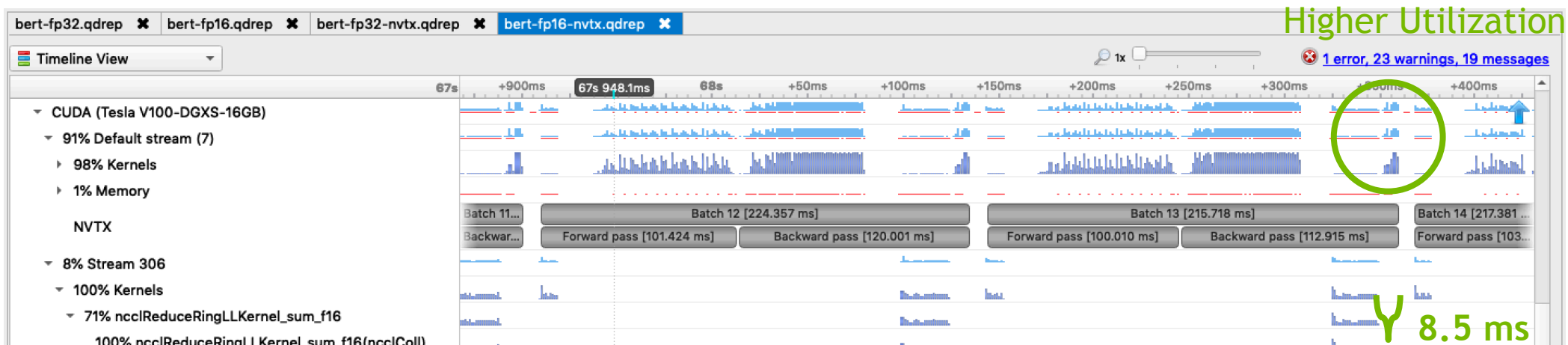
4 V100 GPUs w/ NVLINK, Batch size: 32, max_seq_length: 512

APEX OPTIMIZER OPTIMIZATION

FP32



FP16

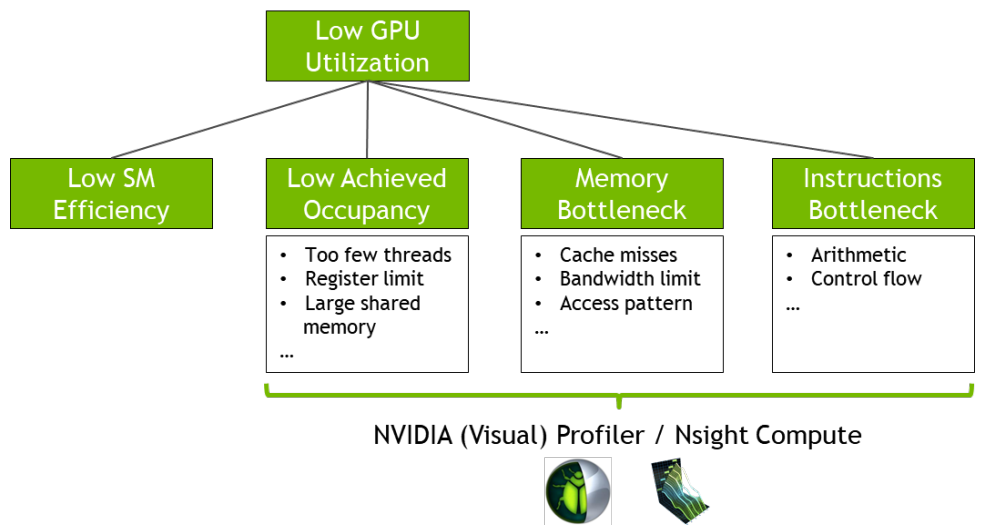


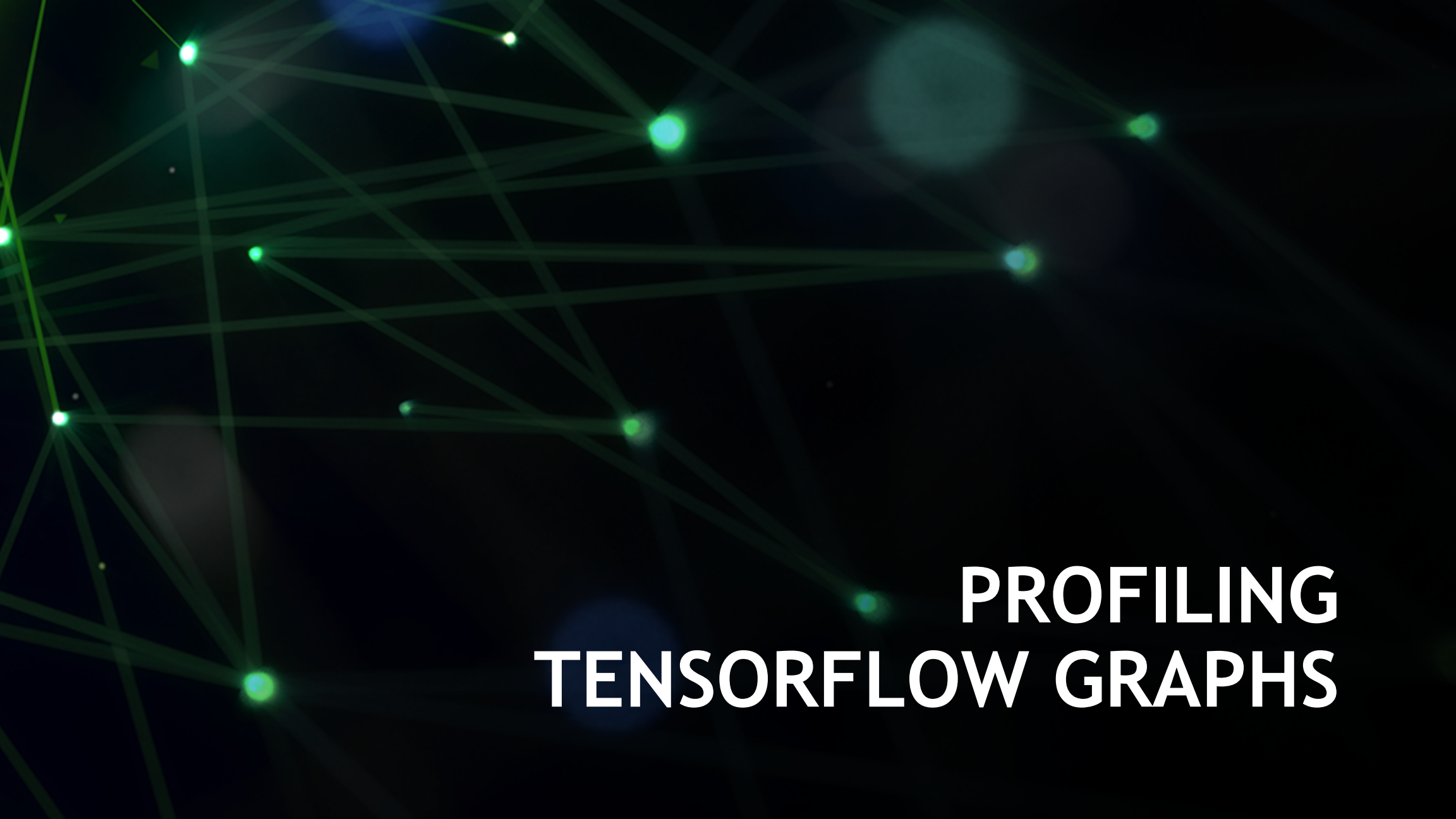
APEX/csrc/adam_cuda_kernel()

PROFILING GPU APPLICATION

Reminding

- You can investigate the performance bottleneck in each layer
- NVIDIA provides such inefficient layers / graphs optimization support
  - For example,
    - Like this, adam_cuda_kernel in APEX
    - cudnnMultiHeadAttnForward() in cuDNN
    - BERT speed-up




PROFILING
TENSORFLOW GRAPHS

NVTX WITH TENSORFLOW

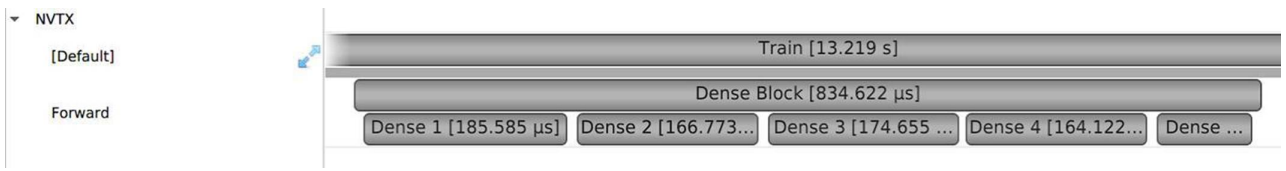
- The TF_DISABLE_NVTX_RANGES variable enables and disables NVTX ranges in TensorFlow
  - NVTX collect is enabled by default
  - Available in NGC Container
- Having NVTX plugin for TF
  - Import nvtx plugin
  - Annotation in python code
  - Get data from through Profiler

```python
import tensorflow as tf

import nvtx.plugins.tf as nvtx_tf

# Option 1: use decorators
@nvtx_tf.layers.trace(message="Dense Block", domain_name="Forward",
                      grad_domain_name="Gradient")
def dense_block(net):
    net = tf.layers.dense(net, 1, activation=tf.nn.relu, name='dense_1')
    net = tf.layers.dense(net, 64, activation=tf.nn.relu, name='dense_2')
    net = tf.layers.dense(net, 128, activation=tf.nn.relu, name='dense_3')
    net = tf.layers.dense(net, 256, activation=tf.nn.relu, name='dense_4')
    net = tf.layers.dense(net, 512, activation=tf.nn.relu, name='dense_5')
    return net


def build_model(inputs)

    x = dense_block(inputs)

    # Option 2: wrap parts of your graph with nvtx layers
    x, nvtx_context = nvtx_tf.layers.start(x, message="logits-softmax")
    x = tf.layers.dense(x, 512, activation=None, name='logits')
    x = tf.nn.softmax(x, name='probs')
    x = nvtx_tf.layers.end(x, nvtx_context)

    return x
```

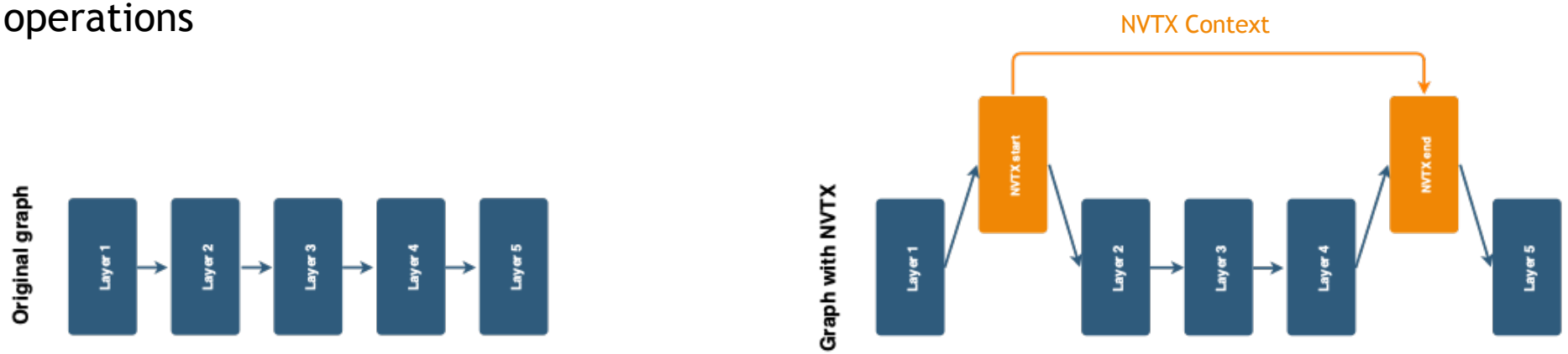
NVTX PLUGINS FOR DEEP LEARNING

Profiling TensorFlow for the graphs

- Allows users to add their own NVIDIA Tools Extension (NVTX) events and time ranges to a TensorFlow graph



- Ranges are added by wrapping regions of the computation graph with start and end operations



SETUP

Install NVTX plugins

```
pip install nvtx-plugins-tf
```

No need to install this plugin since NGC 19.08

HOW TO USE

- Adding Markers

```
import nvtx.plugins.tf as nvtx_tf

x, nvtx_context = nvtx_tf.ops.start(x, message='Dense 1-2', \
        domain_name='Forward', grad_domain_name='Gradient', enabled=ENABLE_NVTX, trainable=True)
x = tf.layers.dense(x, 1000, activation=tf.nn.relu, name='dense_1')
x = tf.layers.dense(x, 1000, activation=tf.nn.relu, name='dense_2')
x = nvtx_tf.ops.end(x, nvtx_context)
x = tf.layers.dense(x, 1000, activation=tf.nn.relu, name='dense_3')
```

- Function Detector

```
import nvtx.plugins.tf as nvtx_tf

@nvtx_tf.ops.trace(message='Dense Block', domain_name='Forward',
        grad_domain_name='Gradient', enabled=ENABLE_NVTX, trainable=True)
def dense_block(x):
x = tf.layers.dense(x, 1000, activation=tf.nn.relu, name='dense_1')
x = tf.layers.dense(x, 1000, activation=tf.nn.relu, name='dense_2')
return x
```

WORKING WITH SESSION/TF.ESTIMATOR

```
from nvtx.plugins.tf.estimator import NVTXHook

nvtx_callback = NVTXHook(skip_n_steps=1, name='Train')
training_hooks=[]
training_hooks.append(nvtx_callback)
```

```
with tf.train.MonitoredSession(hooks=[nvtx_callback]) as sess:
```

```
tf.estimator.Estimator(hooks=training_hooks, ...)
```

PROFIING MPI PROCESS

- To profile everything, putting the data into one file

```
nsys [nsys options] mpirun [mpi options]
```

- To profile everything, putting the data into each node into a separated file

```
mpirun [mpi options] nsys profile [nsys options]
```

COMMAND EXAMPLES

NGC TensorFlow ResNet50-1.5

- 1 GPU profiling

```
nsys profile -y 40 -d 20 -f true -w true -o tf_amp \
     -t osrt,nvtx,cuda,cublas,cudnn \
     python ./main.py --mode=training_benchmark --warmup_steps 200 \
     --num_iter 500 --iter_unit batch --results_dir=results --batch_size 64
```

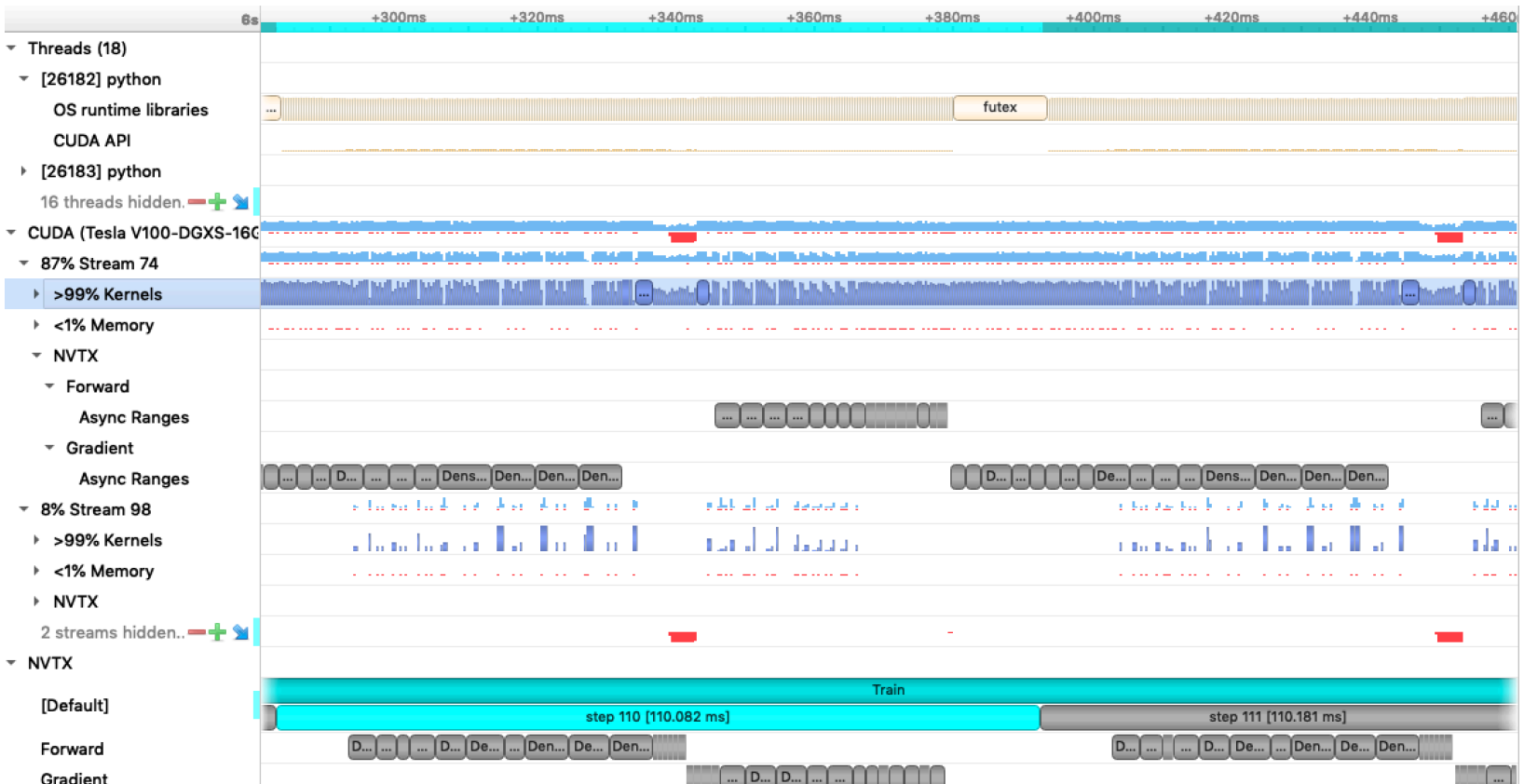
- 4 GPU profiling

```
nsys profile -y 40 -d 20 -f true -w true -o tf_amp_4gpu \
     -t osrt,nvtx,cuda,cublas,cudnn \
     mpiexec --allow-run-as-root --bind-to socket -np 4 \
     python ./main.py --mode=training_benchmark --warmup_steps 200 \
     --num_iter 500 --iter_unit batch --results_dir=results --batch_size 64
```

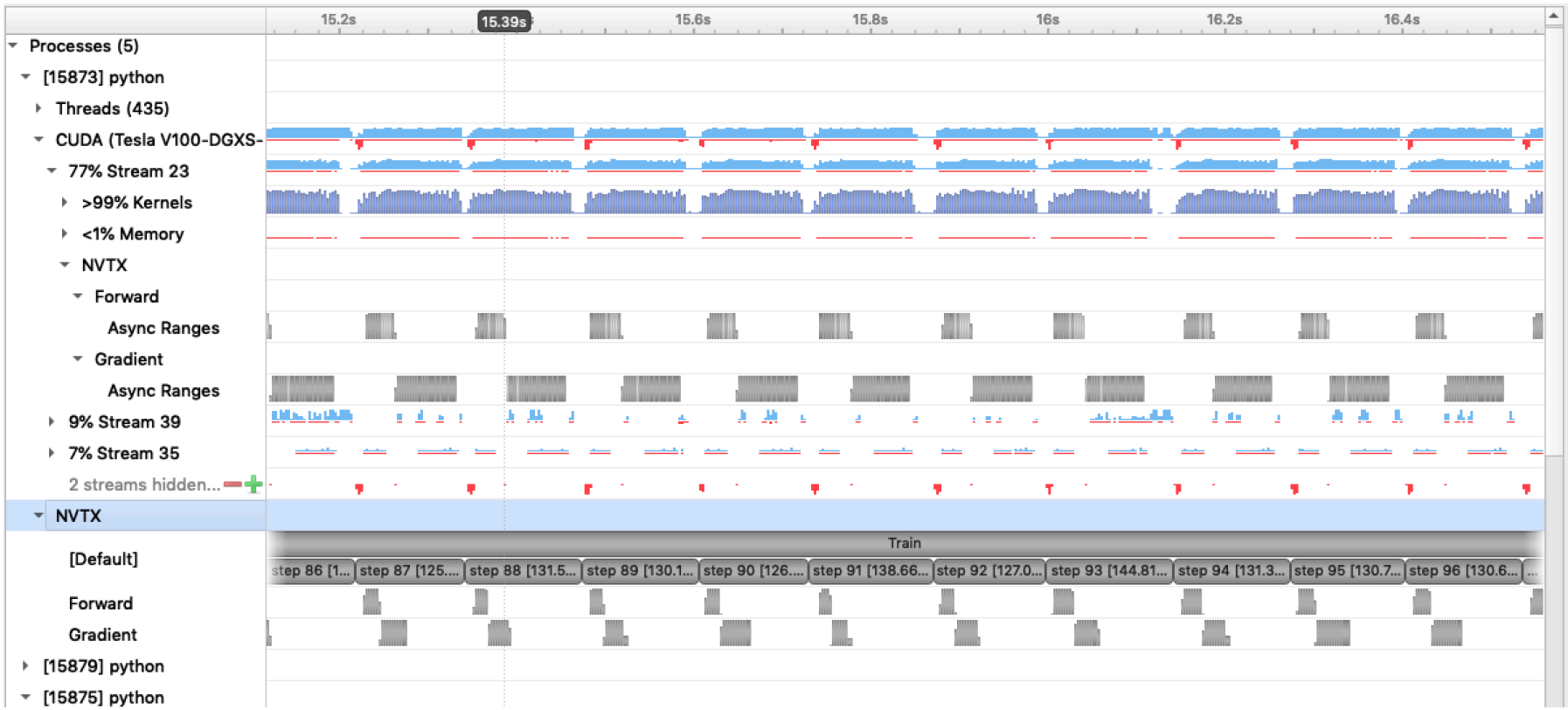
TF WITH NVTX


5s
5s 726.6ms
+750ms
+800ms
+850ms
+900ms
+950ms
Processes (2)
[25631] python
Threads (17)
[25817] python
OS runtime libraries
CUDA API
[25818] python
15 threads hidden...
CUDA (Tesla V100-DGXS-16GB, 0000:07:00.0
84% Stream 74
>99% Kernels
<1% Memory
NVTX
Forward
Async Ranges
Gradient
Async Ranges
12% Stream 98
2% Stream 75
<1% Other streams (3)
NVTX
[Default]
Train
step 31 [186....
step 32 [187.019 ms]
step 33 [186.622 ms]
Forward
Gradient
[25744] orted

AMP ENABLED


6s
+300ms
+320ms
+340ms
+360ms
+380ms
+400ms
+420ms
+440ms
Threads (18)
[26182] python
OS runtime libraries
futex
CUDA API
[26183] python
16 threads hidden.
CUDA (Tesla V100-DGXS-16
87% Stream 74
>99% Kernels
<1% Memory
NVTX
Forward
Async Ranges
Gradient
Async Ranges
8% Stream 98
>99% Kernels
<1% Memory
NVTX
2 streams hidden..
NVTX
Train
[Default]
step 110 [110.082 ms]
step 111 [110.181 ms]
Forward
Gradient

MULTI-GPU


15.2s
15.39s
15.6s
15.8s
16s
16.2s
16.4s
Processes (5)
[15873] python
Threads (435)
CUDA (Tesla V100-DGXS-
77% Stream 23
>99% Kernels
<1% Memory
NVTX
Forward
Async Ranges
Gradient
Async Ranges
9% Stream 39
7% Stream 35
2 streams hidden...
NVTX
Train
[Default]
step 86 [1...
step 87 [125....
step 88 [131.5...
step 89 [130.1...
step 90 [126....
step 91 [138.66...
step 92 [127.0...
step 93 [144.81...
step 94 [131.3...
step 95 [130.7...
step 96 [130.6...
Forward
Gradient
[15879] python
[15875] python


NVIDIA