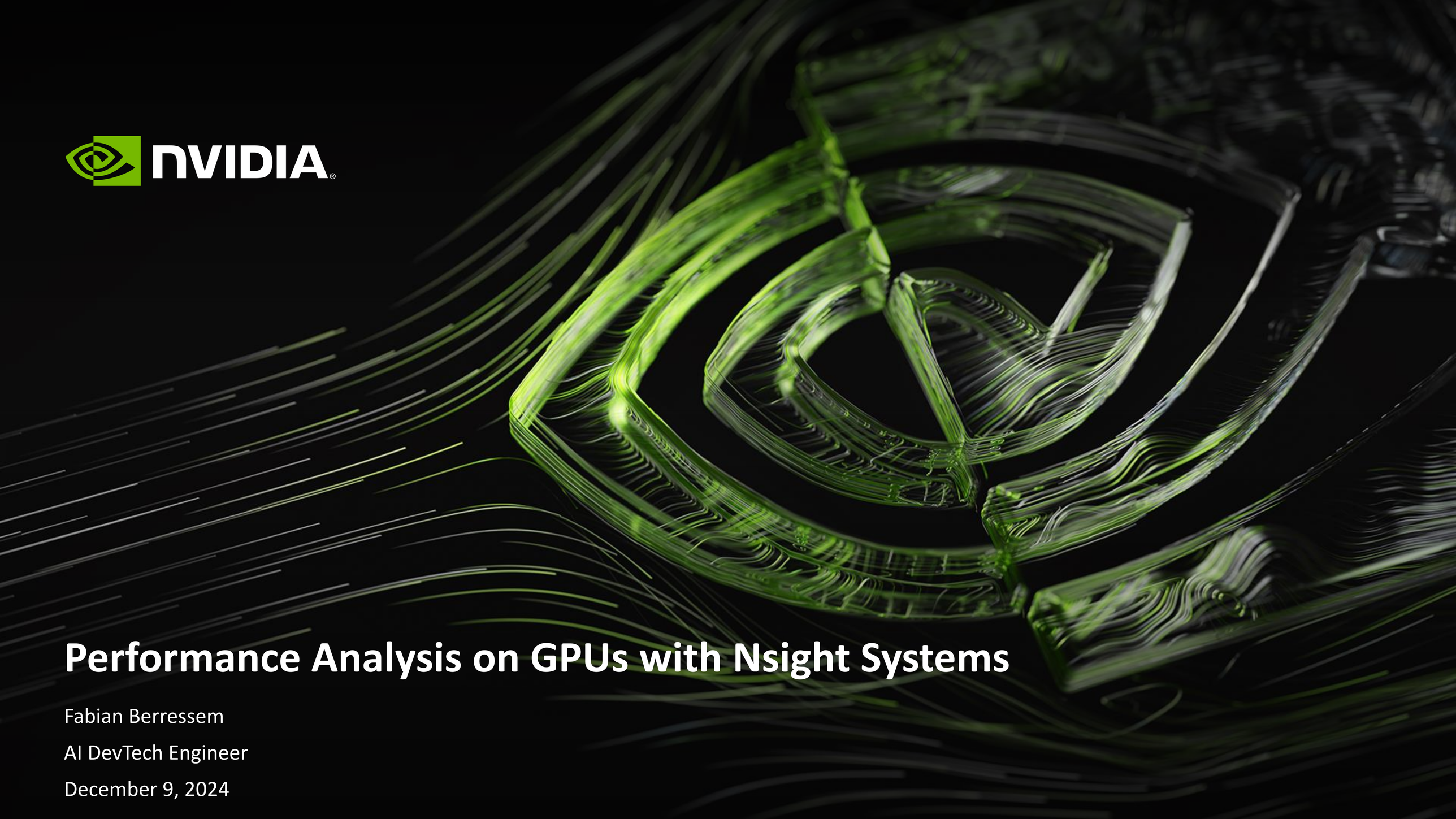


Performance Analysis on GPUs with Nsight Systems

Fabian Berressem

AI DevTech Engineer

December 9, 2024

Agenda

- Nsight Systems
- DLFW Extensions
- Use Case: Optimizing a PyTorch Workload
- [Nsight Compute]

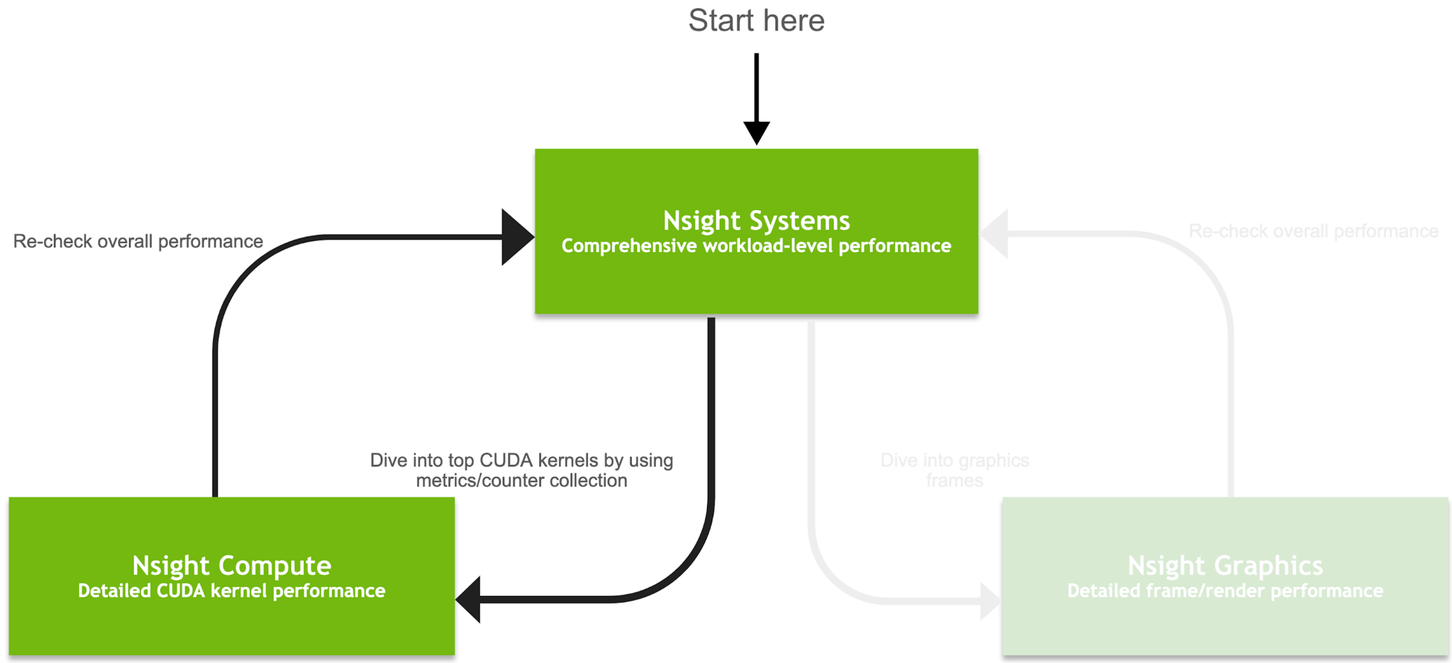
Nsight Product Family

Optimization Workflow

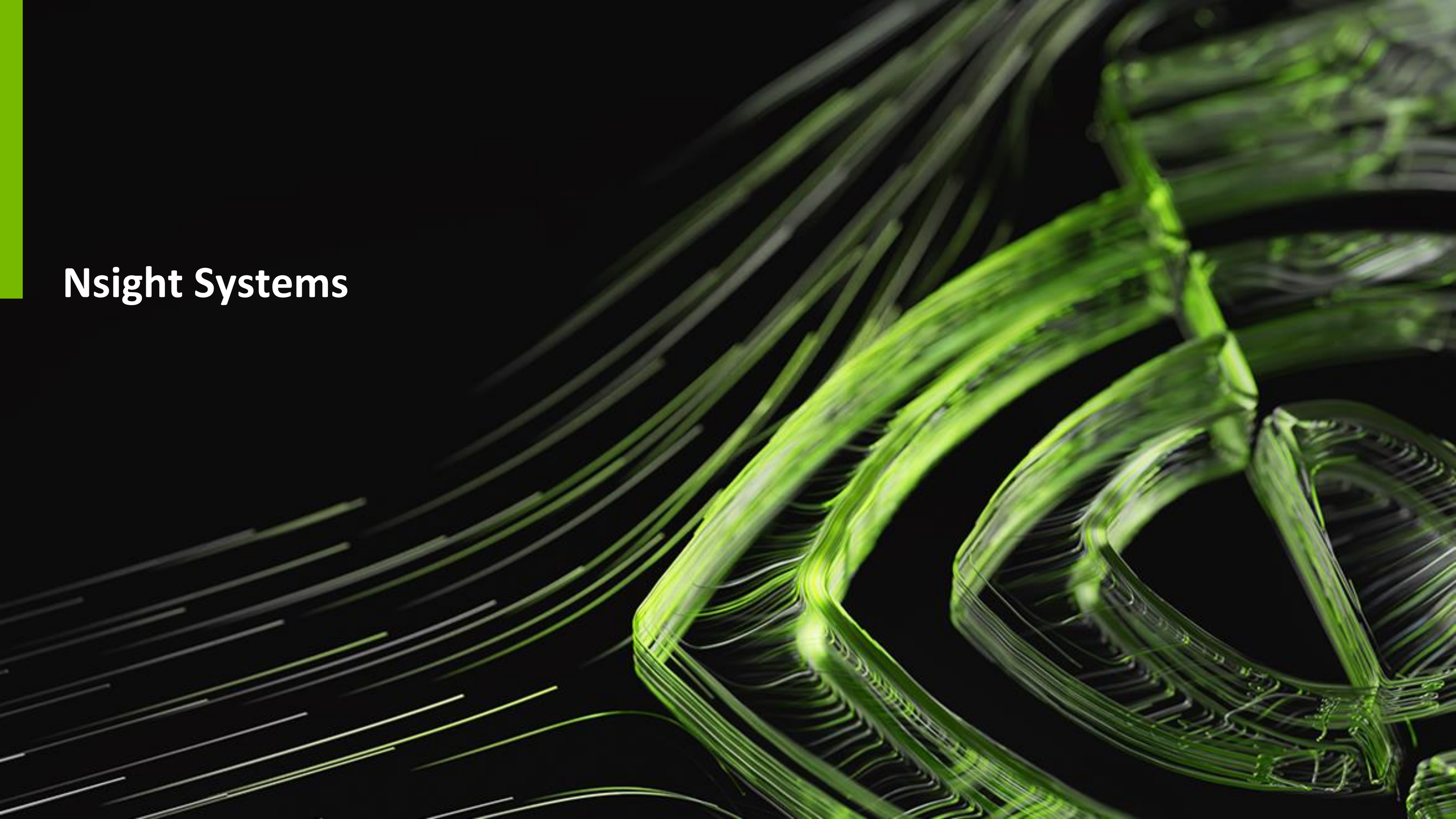
Nsight Systems
Analyze application algorithm system-wide

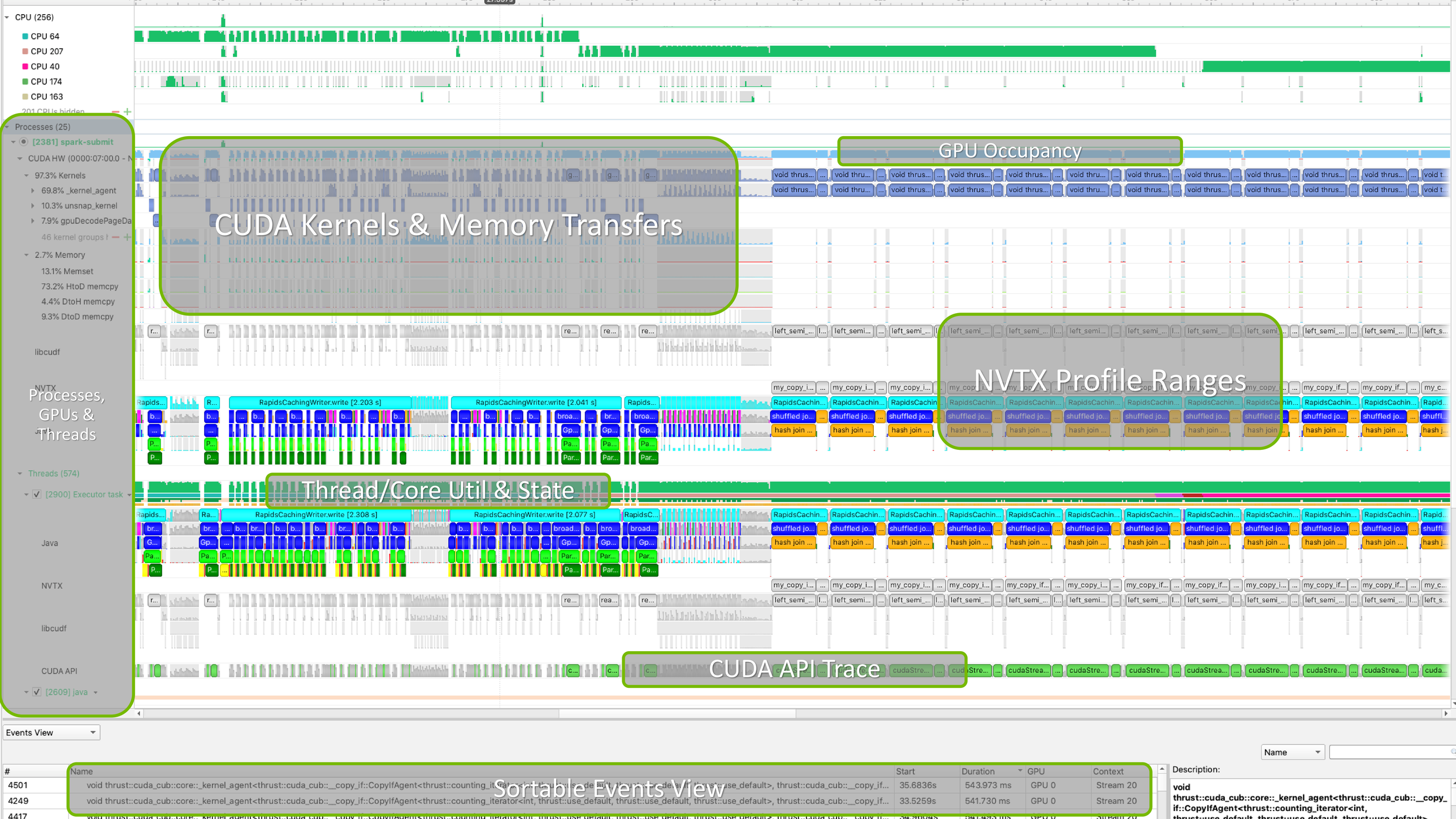
Nsight Compute
Debug/optimize CUDA kernel

Nsight Graphics
Debug/optimize graphics workloads



Nsight Systems




CPU (256)
CPU 64
CPU 207
CPU 40
CPU 174
CPU 163
Processes (25)
[2381] spark-submit
CUDA HW (0000:07:00.0 - N
97.3% Kernels
69.8% _kernel_agent
10.3% unsnap_kernel
7.9% gpuDecodePageDa
46 kernel groups t
2.7% Memory
13.1% Memset
73.2% HtoD memcpy
4.4% DtoH memcpy
9.3% DtoD memcpy
libcudf
NVTX
Processes, GPUs & Threads
CUDA Kernels & Memory Transfers
GPU Occupancy
NVTX Profile Ranges
RapidsCachingWriter.write [2.203 s]
RapidsCachingWriter.write [2.041 s]
Threads (574)
[2900] Executor task
Thread/Core Util & State
RapidsCachingWriter.write [2.308 s]
RapidsCachingWriter.write [2.077 s]
Java
NVTX
libcudf
CUDA API
CUDA API Trace
[2609] java
Events View
Name
#
Name
Start
Duration
GPU
Context
4501
4249
Sortable Events View
Description:

Nsight Systems

CLI - Example

```
$ 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 train.py                    ← Execution command
```

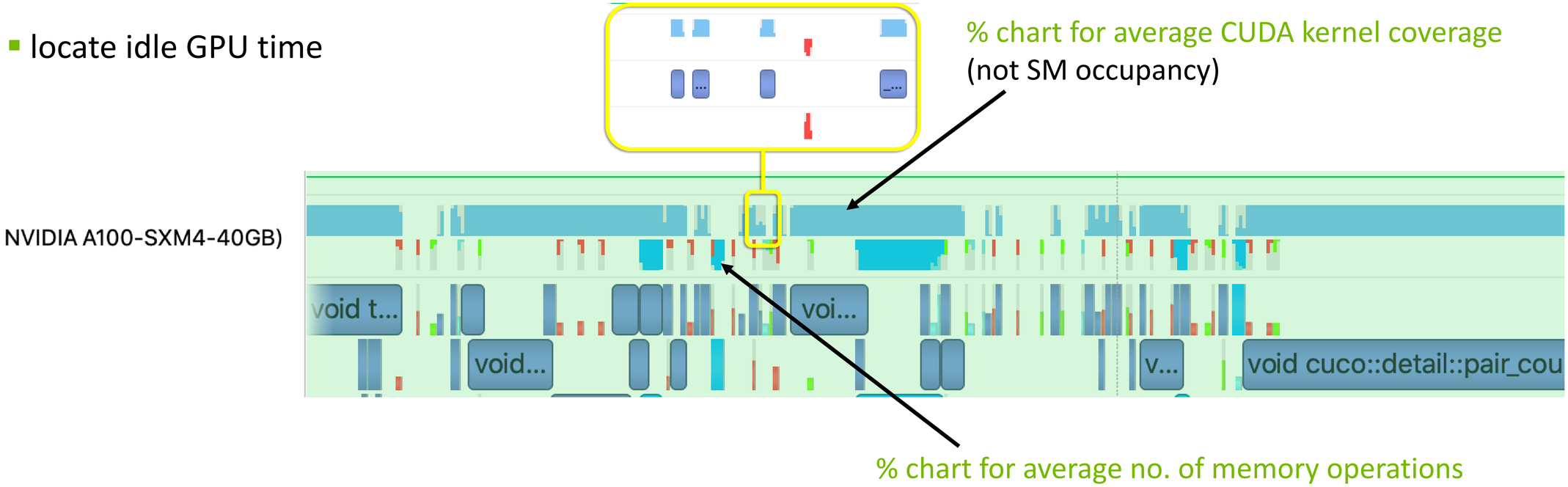
cuda - GPU kernel
osrt - OS runtime (e.g pread, pthread, etc)
nvtx - NVIDIA Tools Extension
cudnn - CUDA Deep NN library
cublas - CUDA BLAS library

https://docs.nvidia.com/nsight-systems/UserGuide/#cli-profiling

Nsight Systems

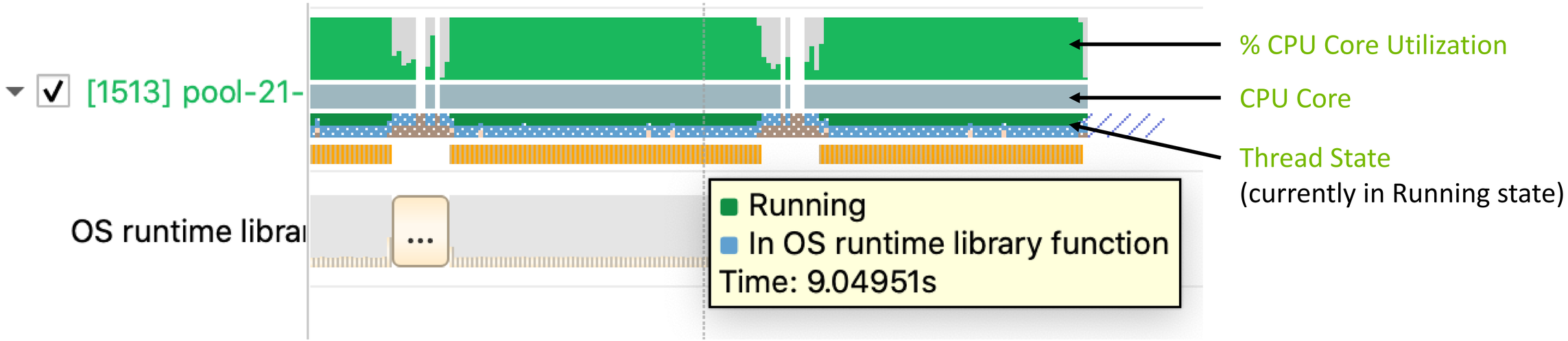
GPU Workload - A Closer Look

- check trace of GPU activity
- locate idle GPU time



Nsight Systems

CPU Thread Activities



Nsight Systems

CLI - Extended Example

```
$ nsys profile ...                          \
               --cuda-memory-usage true \
               --gpu-metrics-device all \
               --nic-metrics true       \
               python train.py
```

https://docs.nvidia.com/nsight-systems/UserGuide/#cli-profiling


GPU (0000:45:00.0 - NVIDIA
GPU Metrics [10 kHz]
activity
memory
GPC Clock Frequency
0 to 2.01 GHz
SYS Clock Frequency
0 to 1.67 GHz
GR Active
0 to 100%
SMs Active
0 to 100%
SM Instructions
0 to 100%
SM Warp Occupancy
100%
0
DRAM Bandwidth
0 to 100%
NVLink RX Bandwidth
0 to 100%
NVLink TX Bandwidth
0 to 100%
PCIe Bandwidth
0 to 100%
Processes (35)
[3596] python3
0 to 100%
CUDA HW (0000:45:00.0 -
kernel
memory
Memory usage
0 to 27.8 GiB
Static memory usage
0 to 250 kiB
Local Memory Pool
>99.9% Kernels
21.1% indexFuncLargeIn
16.7% indexSelectLarge
15.6% vectorized_eleme
10.7% CatArrayBatched
8.5% sm90_xmma_gem
5.6% vectorized_layer_
5.5% _scatter_gather_e
5.1% sm90_xmma_gem
4.4% sm90_xmma_gem
4.4% elementwise_kern
2.5% reduce_kernel
GPU Clocks
Instructions / Occupancy
(incl. TensorCore)
Bandwidths DRAM/NVLink/PCIE
Allocated Memory
CUDA Kernels & Memory Transfers

Nsight Systems

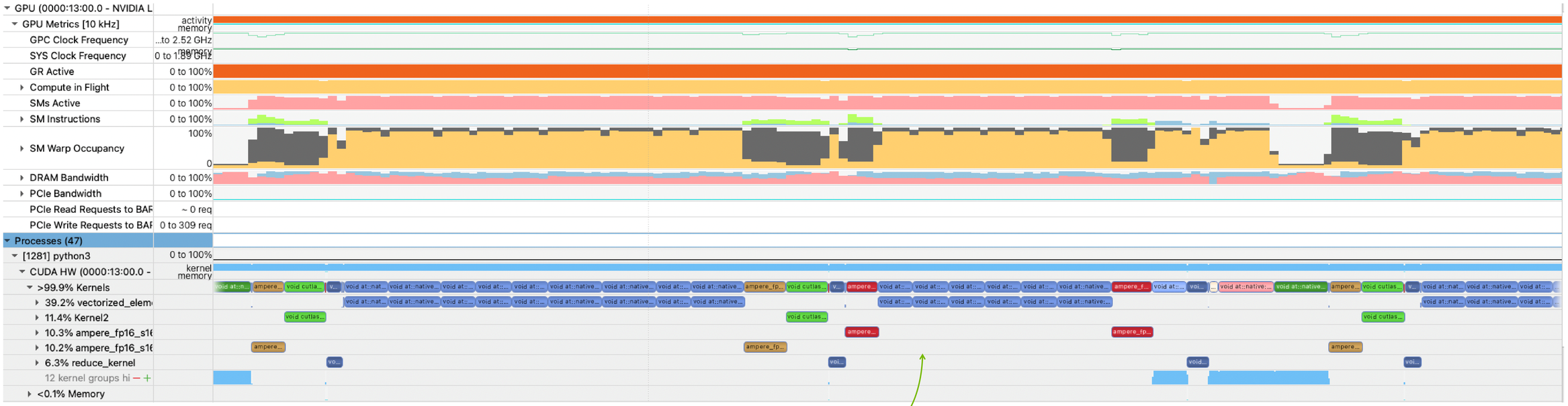
A PyTorch Example

```
def my_naive_silu_activation(x):
   return x * (1.0 / (1.0 + torch.exp(-x))

class MySimpleNeuralNetwork(nn.Module):
      def __init__(self):
            super().__init__()
            self.lin1 = nn.Linear(512, 512)
            self.lin2 = nn.Linear(512, 512)
            self.norm = nn.LayerNorm(512)

      def forward(self, x: torch:Tensor):
            x = self.lin1(x)
            x = my_naive_silu_activation(x)
            x = self.lin2(x)
            x = self.norm(x)
            return x
```

```
def main():
   x = torch.randn(...)

   model = MySimpleNeuralNetwork()
   model = model.to(
      device=”cuda",
      dtype=torch.half
   )

   for _ in range(10):
      x = model(x)
      loss = x.sum()
      loss.backward()
```

Nsight Systems

A PyTorch Example – A First Profile



just a sequence of kernels
where are they stemming from?
what is forward, what is backward?

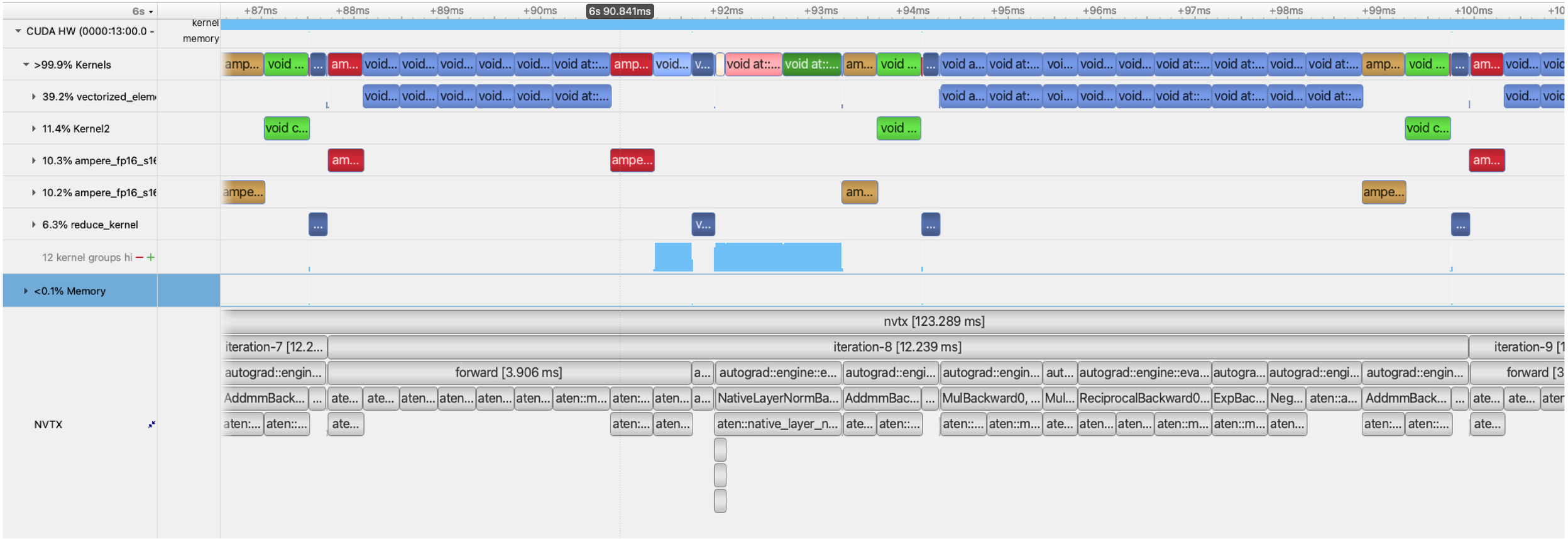
Nsight Systems

A PyTorch Example – NVTX Annotations

```
def main():
   x = torch.randn(...)

   # automatic emission of nvtx ranges for each module
      with torch.autograd.profiler.emit_nvtx():
      model = MySimpleNeuralNetwork()
      model = model.to(
         device=”cuda",
         dtype=torch.half
      )

      for _ in range(10):
         # manual ranges via push/pop
         torch.cuda.nvtx.range_push(f”iter-{it}”)
         torch.cuda.nvtx.range_push(“forward”)
         x = model(x)
         torch.cuda.nvtx.range_pop()
         loss = x.sum()
         loss.backward()
                  torch.cuda.nvtx.range_pop()
```

Nsight Systems

A PyTorch Example – NVTX Annotations



NVIDIA

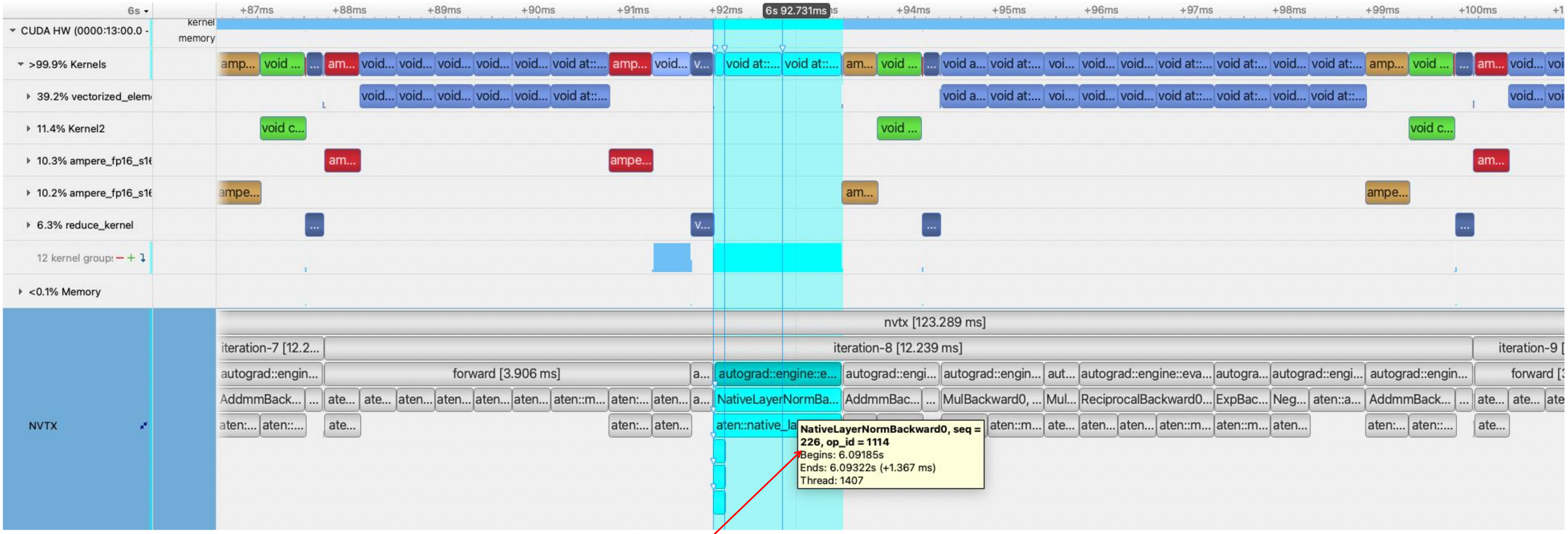
Nsight Systems

A PyTorch Example – NVTX Annotations

correlate forward and backward kernels by matching sequence IDs

Nsight Systems

A PyTorch Example – NVTX Annotations



correlate forward and backward kernels by matching sequence IDs

NVIDIA

Nsight Systems

Don’t want GUI? Use ‘**stats**’

> nsys stats baseline.nsys-rep

** CUDA GPU Kernel Summary (cuda_gpu_kern_sum):

Time (%)	Total Time (ns)	Instances	Avg (ns)	Med (ns)	Min (ns)	Max (ns)	StdDev (ns)	Name
16.2	66952090	120	557934.1	599714.0	399233	630402	76713.4	void at::native::vectorized_elementwise_kernel<(int)4, at::native::BinaryFunctor<c10::Half, c10::Ha…
11.4	47258303	100	472583.0	473729.0	457762	481346	5047.6	void cutlass::Kernel2<cutlass_80_tensorop_f16_s16816gemm_relu_f16_64x256_32x4_nt_align8>(T1::Params)
10.3	42571625	100	425716.3	420161.5	373601	487490	48382.9	ampere_fp16_s1688gemm_fp16_128x128_ldg8_relu_f2f_tn
10.2	42131662	100	421316.6	420593.0	377698	483041	33937.1	ampere_fp16_s1688gemm_fp16_128x128_ldg8_f2f_nn
5.8	23991470	60	399857.8	398913.5	383650	413346	7701.5	void at::native::vectorized_elementwise_kernel<(int)4, at::native::neg_kernel_cuda(at::TensorIterat…
4.5	18723388	30	624112.9	623505.5	617154	634946	4653.7	void at::native::<unnamed>::GammaBetaBackwardCUDAKernel_32x32<c10::Half, float>(long, long, const T…
4.5	18476860	30	615895.3	616626.0	606530	623042	4244.7	void at::native::<unnamed>::layer_norm_grad_input_kernel_vectorized<c10::Half, float>(const T1 *, c…
3.8	15854260	40	396356.5	398673.5	377249	410913	12517.7	void at::native::vectorized_elementwise_kernel<(int)4, at::native::AUnaryFunctor<c10::Half, c10::Ha…
3.6	14732714	80	184158.9	184224.0	182081	186048	767.4	void at::native::reduce_kernel<(int)128, (int)4, at::native::ReduceOp<c10::Half, at::native::func_w…
3.4	13953330	30	465111.0	398977.0	380033	619554	105516.1	triton__0d1d2d3d4d5d6d7d
3.0	12499140	290	43100.5	1536.0	1056	606466	151719.8	void at::native::vectorized_elementwise_kernel<(int)4, at::native::CUDAFunctor_add<c10::Half>, at::…
2.9	11982314	30	399410.5	399650.0	393313	403521	2474.1	void at::native::<unnamed>::vectorized_layer_norm_kernel<c10::Half, float>(int, T2, const T1 *, con…
2.7	11353705	50	227074.1	227537.0	220673	230592	2877.9	void at::native::reduce_kernel<(int)512, (int)1, at::native::ReduceOp<c10::Half, at::native::func_w…
2.5	10125601	20	506280.1	506178.0	501826	512002	2780.3	void at::native::vectorized_elementwise_kernel<(int)4, at::native::<unnamed>::silu_backward_kernel(…
2.2	8955198	20	447759.9	450242.0	389569	503713	55864.5	triton__0d1d2d
2.0	8218880	60	136981.3	1152.0	1120	414849	193689.8	void at::native::vectorized_elementwise_kernel<(int)4, at::native::CUDAFunctorOnSelf_add<c10::Half>…
2.0	8146104	20	407305.2	407185.0	402786	414625	3272.8	void at::native::vectorized_elementwise_kernel<(int)4, at::native::reciprocal_kernel_cuda(at::Tenso…
2.0	8124186	20	406209.3	407073.5	369729	414945	9208.7	void at::native::vectorized_elementwise_kernel<(int)4, at::native::exp_kernel_cuda(at::TensorIterat…
1.9	7838075	20	391903.8	391473.0	389025	395906	1924.4	void at::native::vectorized_elementwise_kernel<(int)4, at::native::<unnamed>::silu_kernel(at::Tenso…
1.7	7022873	10	702287.3	702674.5	689827	719331	9521.8	void transformer_engine::layer_norm::ln_bwd_general_kernel<transformer_engine::layer_norm::Kernel_t…
1.5	6310706	50	126214.1	126000.5	124225	128992	1076.4	void at::native::elementwise_kernel<(int)128, (int)4, void at::native::gpu_kernel_impl_nocast<at::n…
1.1	4511727	10	451172.7	452193.0	442306	457602	5517.1	void transformer_engine::layer_norm::ln_fwd_general_kernel<transformer_engine::layer_norm::Kernel_t…
0.7	2694249	60	44904.2	2176.5	1983	131232	61047.9	triton__0d1d2d3d
0.1	281631	100	2816.3	2816.0	2752	2944	35.4	void cublasLt::splitKreduce_kernel<(int)32, (int)16, int, __half, __half, float, __half, (bool)1, (…
0.1	235489	10	23548.9	23552.0	23296	23872	157.1	void transformer_engine::layer_norm::ln_bwd_finalize_general_kernel<__half, float, (unsigned int)4,…
0.0	63649	50	1273.0	1280.0	1119	1440	77.7	void at::native::vectorized_elementwise_kernel<(int)4, at::native::FillFunctor<c10::Half>, at::deta…
0.0	3776	2	1888.0	1888.0	1760	2016	181.0	void at::native::unrolled_elementwise_kernel<at::native::direct_copy_kernel_cuda(at::TensorIterator…
0.0	1696	2	848.0	848.0	832	864	22.6	void at::native::vectorized_elementwise_kernel<(int)4, at::native::FillFunctor<float>, at::detail::…

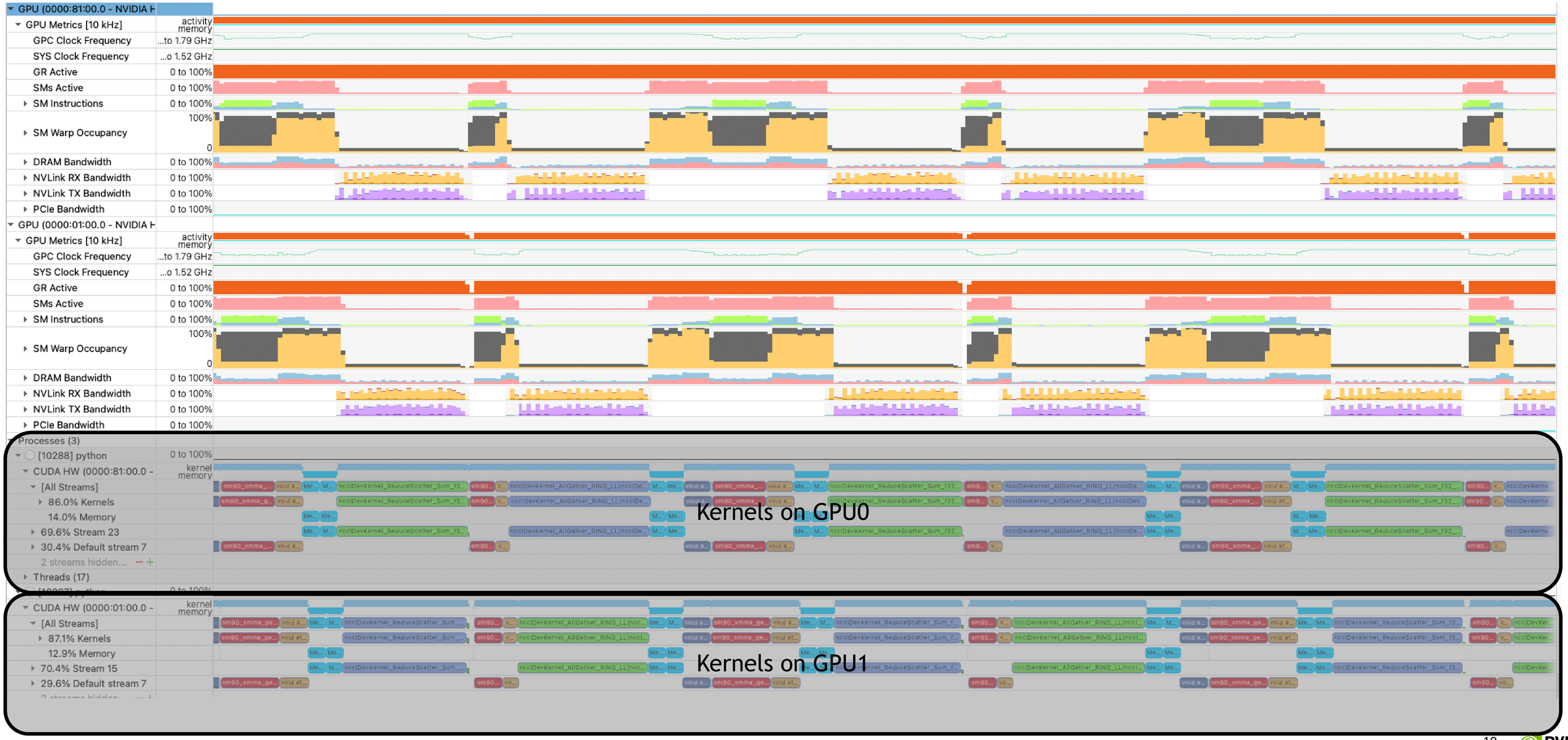
Nsight Systems

Distributed Workload – Single Entrypoint

```
$ nsys profile ... torchrun --nproc-per-node 2 train.py
```

Nsight Systems

Multi-GPU Profile



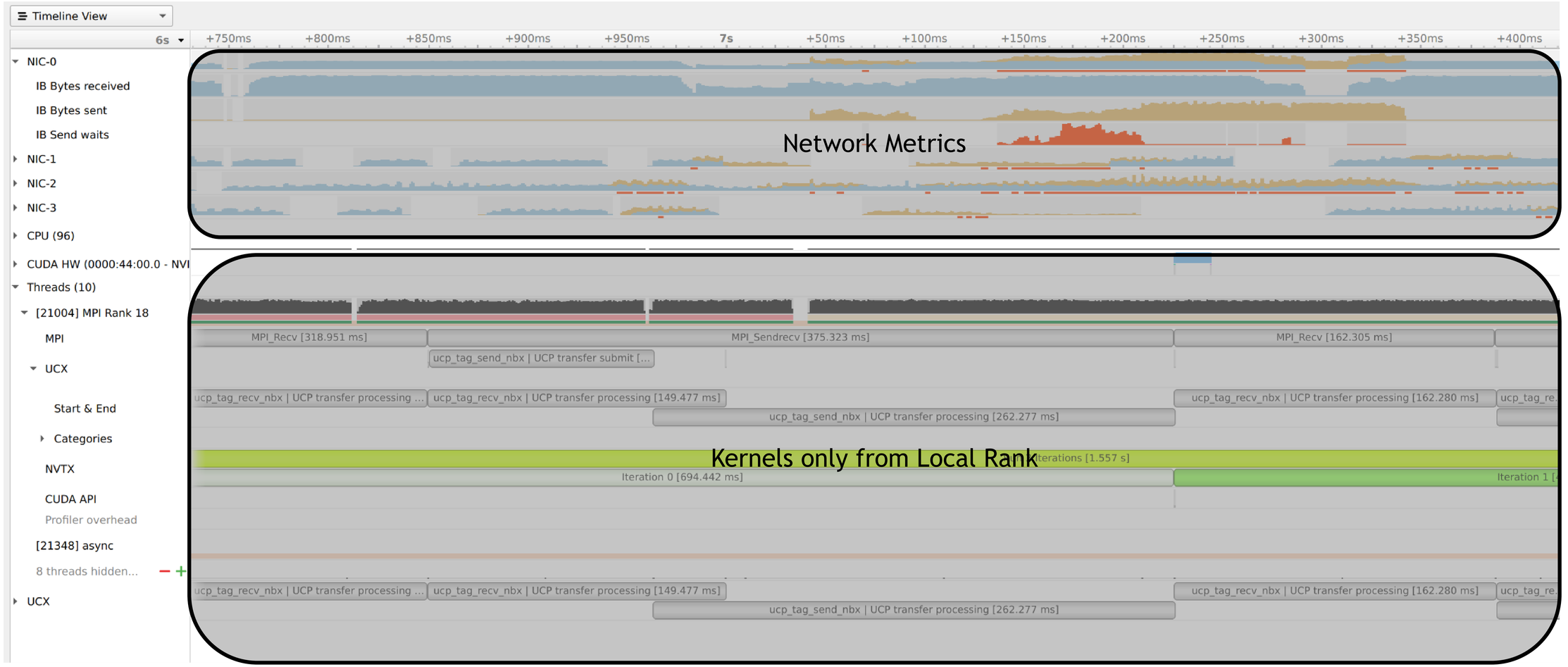
Nsight Systems

Distributed Workload – Many Entrypoints

```
$ mpirun -n 2 nsys profile ... <application>
```

Nsight Systems

Network Analysis – NIC/HCA Metrics Sampling



Nsight Systems

Recently Added Features

- **Remote UI Streaming**

- **Jupyter** Notebook / Lab Extension (profile individual cells)

- **Multi-Node Profiling**
  - most of the times, no single point of entry, i.e. need to profile on each process independently
  - use “recipes” to aggregate information from all these reports
    - `nsys recipe [<args>] <recipe_name> [<recipe_args>]`
    - runtime summaries, utilization heatmaps, communication analysis, and many more

Use Case
Optimizing a PyTorch Workload
But First
Intro in “DLFW Extensions”

DLFW Extensions

Extension Libraries that make Deep Learning FrameWorks faster

- **apex** (https://github.com/NVIDIA/apex)
  - collection of different primitives, unstructured but still occasionally useful
  - historically things like a FusedLayerNorm, a Fused Adam optimizer, AMP, etc. which eventually ended up in PyTorch
  - apex/contrib some fused operations like MHA, Triton kernels e.g. for OpenFold

- **TransformerEngine** (https://docs.nvidia.com/deeplearning/transformer-engine/index.html)
  - mainly layers useful for training transformers in lower precision (FP8 and FP16) like fused attention layers, LayerNorm, fused linear+norm layers and support for model-parallelism
  - wrappers for both JAX and PyTorch

- **cuDNN Frontend**
  - besides using cuDNN as a potential library of many “NN” operations in C++ codes, now also a Python frontend
  - however, often already used by other libraries like TransformerEngine or PyTorch already

- **Other NVIDIA Libraries**
  - RAPIDS as a whole ecosystem of accelerated primitives
  - cuDF (“pandas on the GPU”), cuML (“scikit-learn on the GPU”), XGBoost, cuVS (Vector Search & Clustering), and many more
  - cuPyNumeric (drop-in replacement for NumPy) https://github.com/nv-legate/cupynumeric

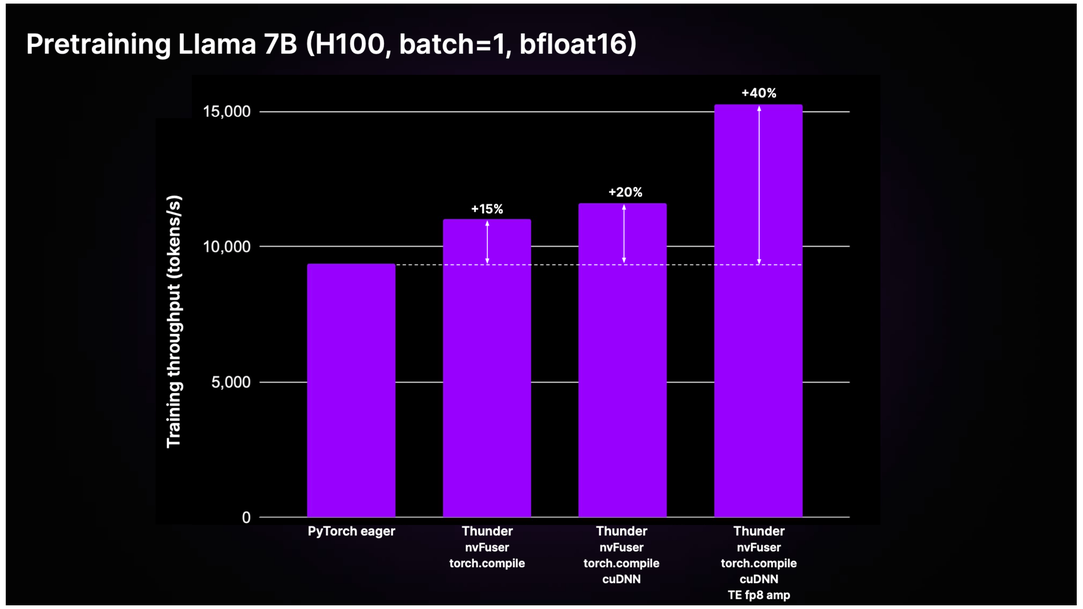
```
import cupynumeric as np  # instead of numpy
```

DLFW Extensions

Just-in-Time Deep Learning Compilers

- **PyTorch** `torch.jit.script / torch.compile`
  - previously, e.g. `torch.jit.script` with nvFuser in the backend
  - now TorchDynamo / TorchInductor in the backend which generates Triton kernels
  - idea: fuse operations together to avoid storing/loading intermediate results, especially useful for memory-bound operations

- **Lightning Thunder**
  - effort by lightning.ai (team behind PyTorch Lightning)
  - don’t use Triton exclusively for code generation, but also include other tools like nvFuser or kernel libraries like cuDNN
  - currently in Alpha Stage



DLFW Extensions

Manual CUDA / PTX Code and JIT Compilers

- **CUDA**
  - always an option ☺ but usually more effort
  - large number of math libraries, like cuDNN, cuFFT, cuTensor, cuSparse, etc,
  - CUTLASS as open-source header-only library for GEMM kernels

- **OpenAI Triton** (https://triton-lang.org/main/index.html)
  - pythonic way of writing lower-level GPU code on a thread-block level
  - JIT compiled

- **nvFuser (**https://github.com/NVIDIA/Fuser)
  - Fusion Code Generator for NVIDIA GPUs
  - Python Frontend and C++ Frontend
  - higher-level language which defines operations to be fused within a FusionDefinition

- **Other**
  - Numba https://numba.pydata.org/numba-doc/latest/cuda/
  - Warp https://github.com/NVIDIA/warp

DLFW Extensions

OpenAI Triton Example

```
BLOCK_SIZE = 512

@jit
def add(X, Y, Z, N):
    pid = program_id(0)
    idx = pid * BLOCK_SIZE + arange(BLOCK_SIZE)
    mask = idx < N
    x = load(X + idx, mask=mask)
    y = load(Y + idx, mask=mask)
    store(Z + idx, x + y, mask=mask)

grid = (ceil_div(N, BLOCK_SIZE),)
add[grid](x, y, z, x.shape[0])
```

DLFW Extensions

nvFuser Example

```
with nvfuser.FusionDefinition() as fd:
    t0 = fd.define_tensor(
        shape=[-1, 1, -1],
        contiguity=[True, None, None],
        dtype=DataType.Float
    )
    t1 = fd.define_tensor(3)
    c0 = fd.define_scalar(3.0)
    t2 = fd.ops.add(t0, t1)
    t3 = fd.ops.mu,(t2, c0)
    t4 = fd.ops.sum(t3, [-1], False, DataType.Float)
    fd.add_output(t4)

torch_out = fd.execute([torch_tensor1, torch_tensor1])[0]
```

Use Case
Optimizing a PyTorch Workload

Optimizing a PyTorch Workload

A PyTorch Example

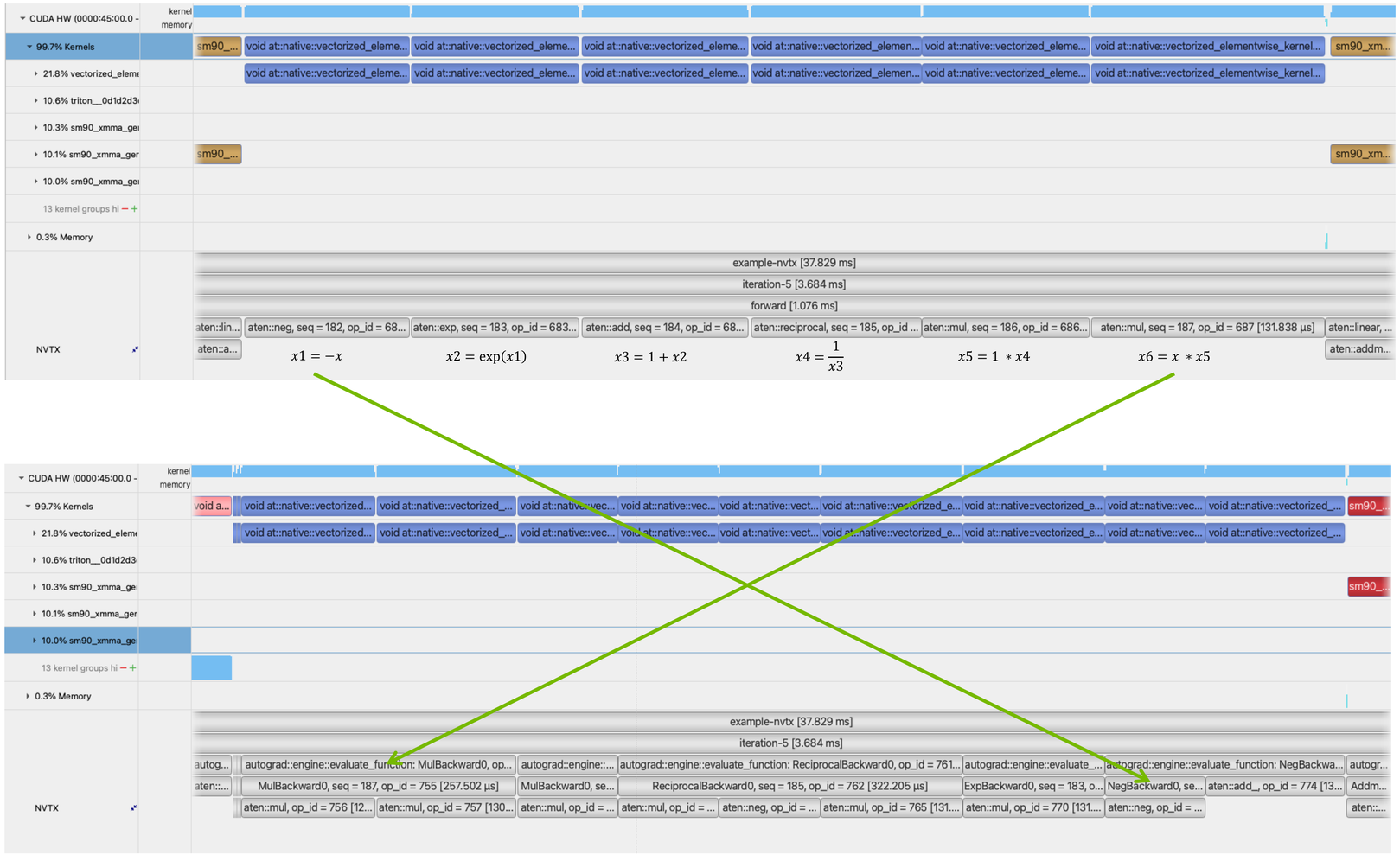
```
def my_naive_silu_activation(x):
   return x * (1.0 / (1.0 + torch.exp(-x))

class MySimpleNeuralNetwork(nn.Module):
      def __init__(self):
            super().__init__()
            self.lin1 = nn.Linear(512, 512)
            self.lin2 = nn.Linear(512, 512)
            self.norm = nn.LayerNorm(512)

      def forward(self, x: torch:Tensor):
            x = self.lin1(x)
            x = my_naive_silu_activation(x)
            x = self.lin2(x)
            x = self.norm(x)
            return x
```

Optimizing a PyTorch Workload

Naïve Sequence of Elementwise Operation

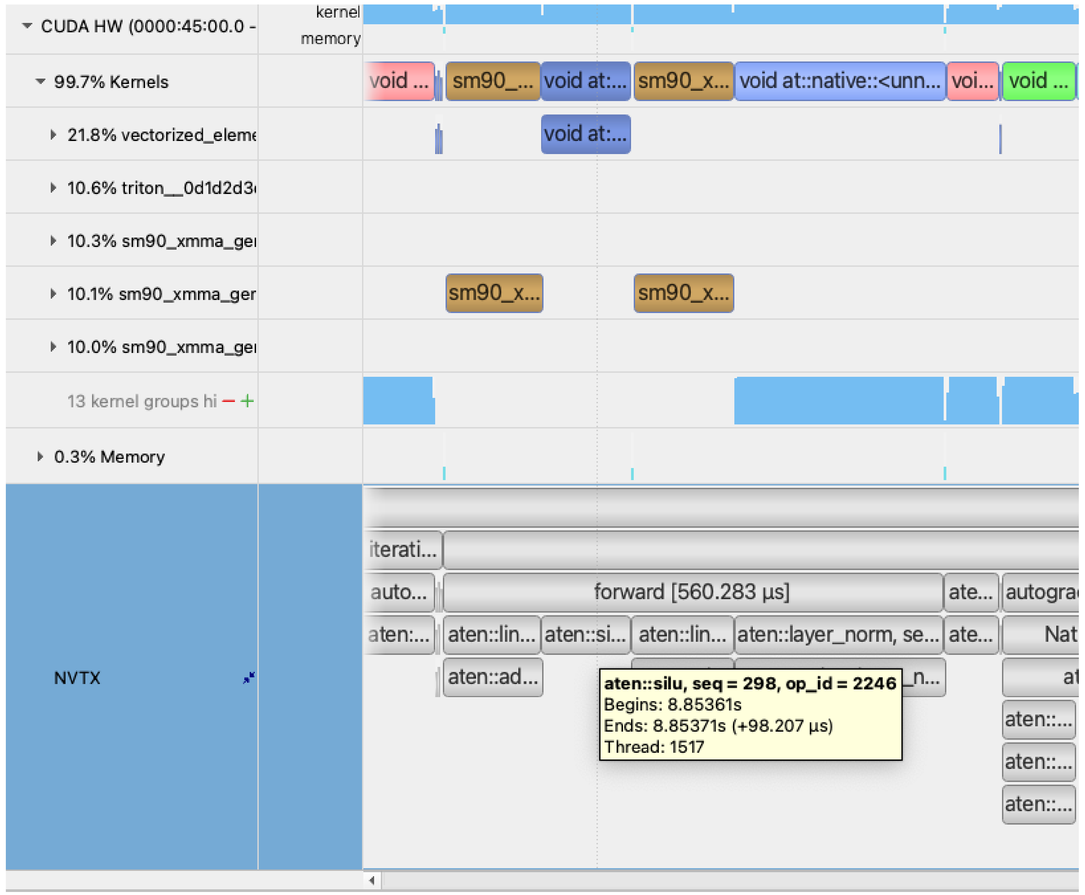
$$SiLU(x) = x * \frac{1}{1 + e^{-x}}$$

$$x1 = -x$$
$$x2 = \exp(x1)$$
$$x3 = 1 + x2$$
$$x4 = \frac{1}{x3}$$
$$x5 = 1 * x4$$
$$x6 = x * x5$$



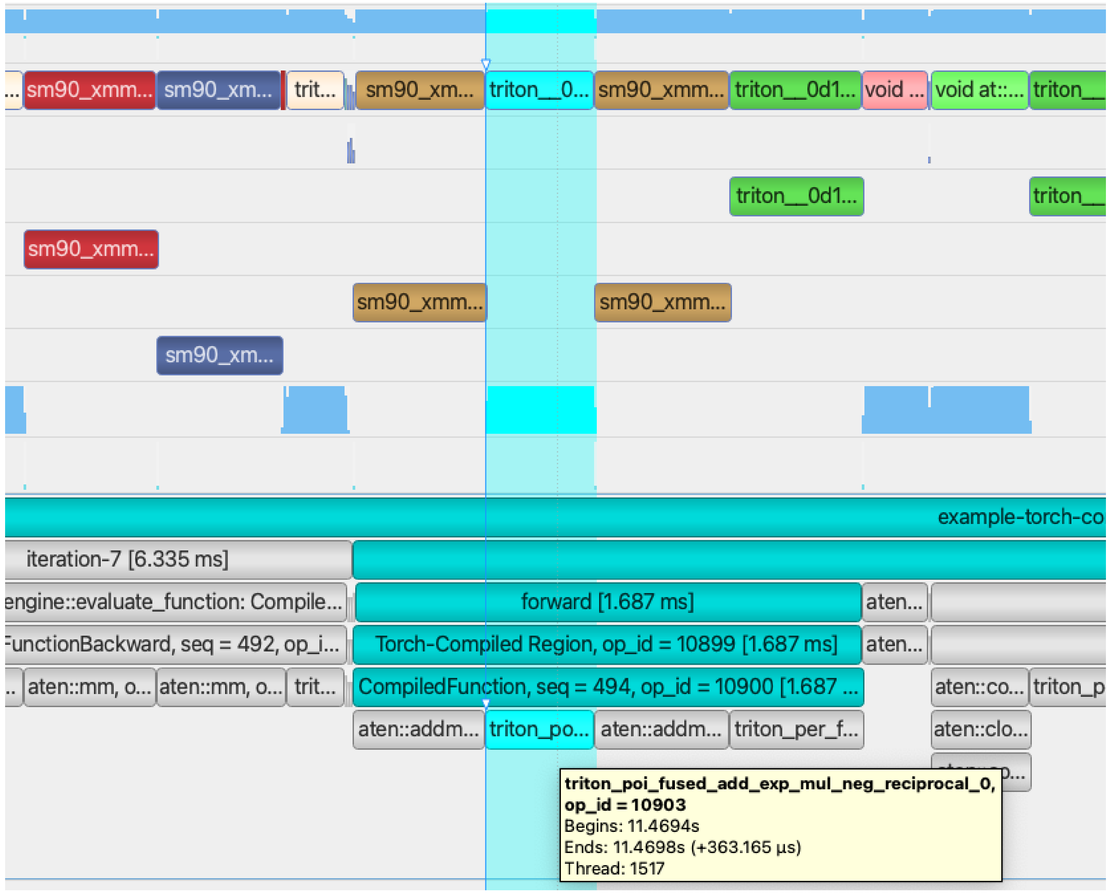
Optimizing a PyTorch Workload

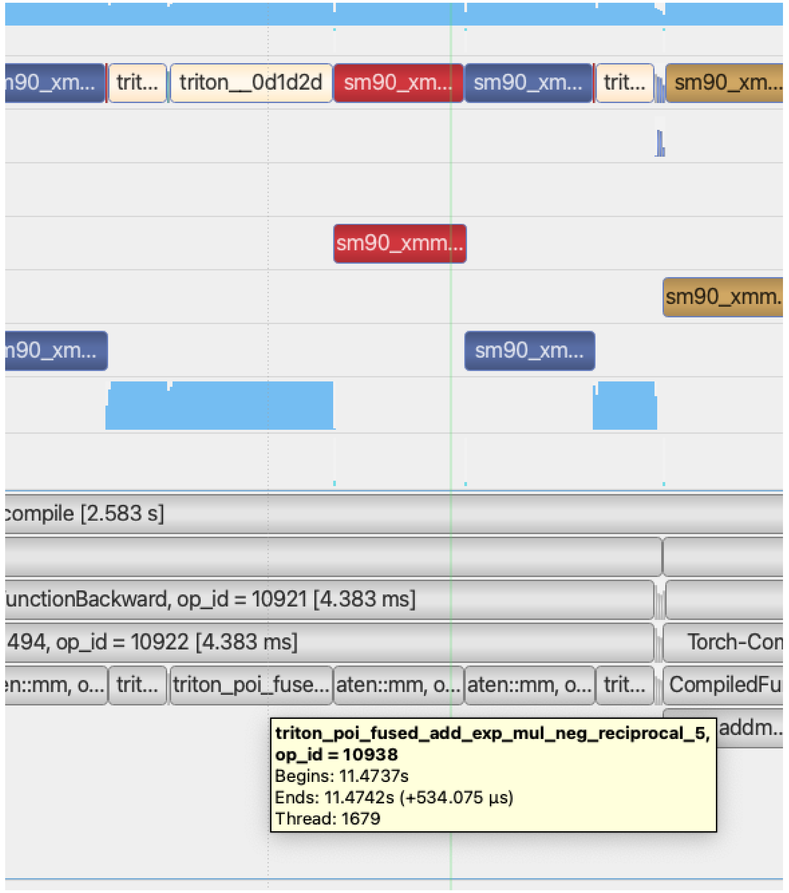
Native “Fused” SiLU Activation in PyTorch



Optimizing a PyTorch Workload

Fused SiLU from torch.compile





Optimizing a PyTorch Workload

SiLU - Summary

Kernel Runtimes [mus]	Forward	Backward
PyTorch – Naïve SiLU	615	1025
PyTorch – Native SiLU	100	128
PyTorch – Naïve+torch.compile	90	128

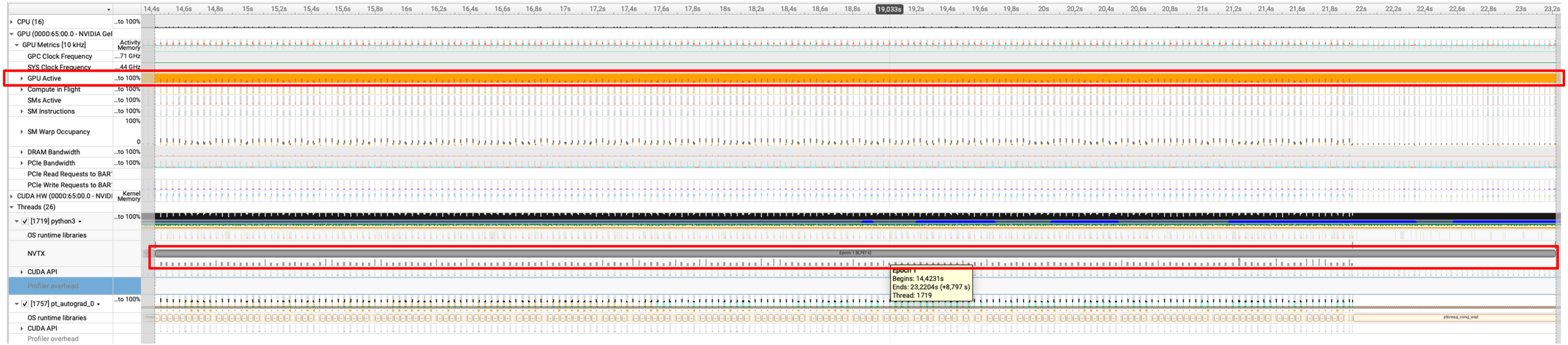
Optimizing a PyTorch Workload

Typical dataloading setup

```
# Download training data from open datasets.
training_data = datasets.CIFAR100(
    root="data",
    train=True,
    download=True,
    transform=ToTensor(),
)

# Download test data from open datasets.
test_data = datasets.CIFAR100(
    root="data",
    train=False,
    download=True,
    transform=ToTensor(),
)

# Create data loaders.
train_dataloader = DataLoader(training_data, batch_size=batch_size, pin_memory=True, num_workers=num_workers)
test_dataloader = DataLoader(test_data, batch_size=batch_size, pin_memory=True, num_workers=num_workers)
```

Optimizing a PyTorch Workload

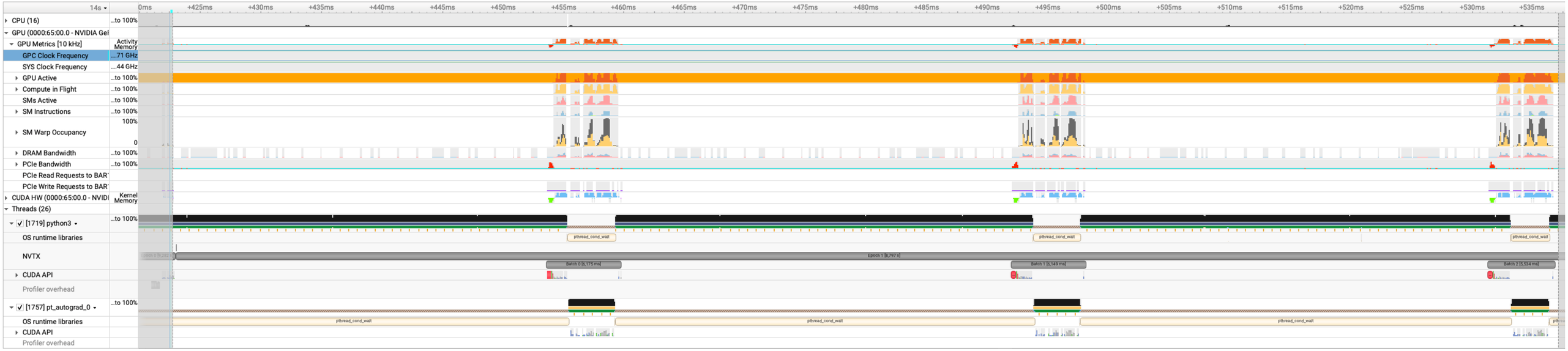
Nsight Systems profile for num_workers=0 (default value)



- GPU barely active
- Computation seems strangely sparse
- Runtime of about 8.8s per epoch

Optimizing a PyTorch Workload

Nsight Systems profile for num_workers=0 (default value)
A more detailed look

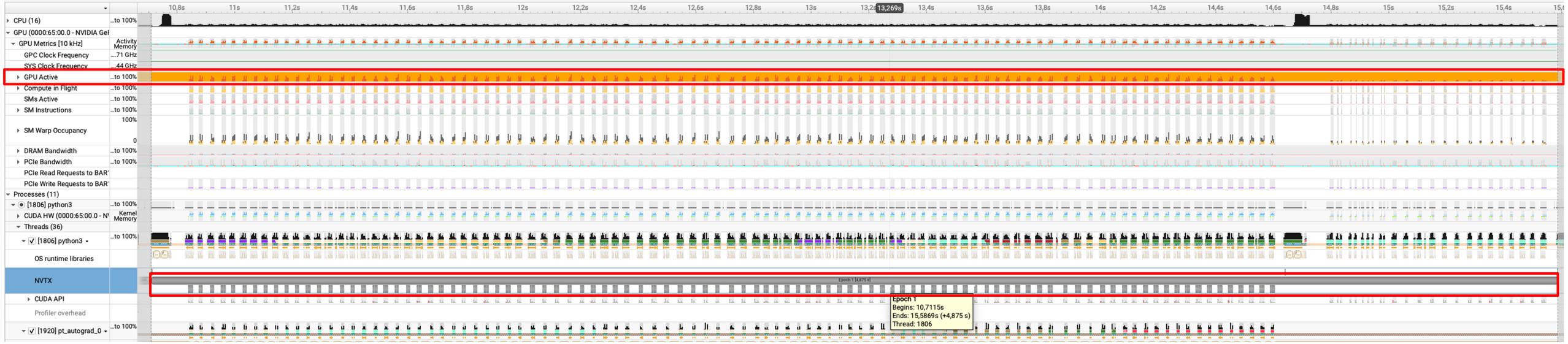


- Huge gaps between batches
- GPU only active during computations in batches, is idle during data loading

We have to keep the GPU busy
-> Increase the number of workers!

Optimizing a PyTorch Workload

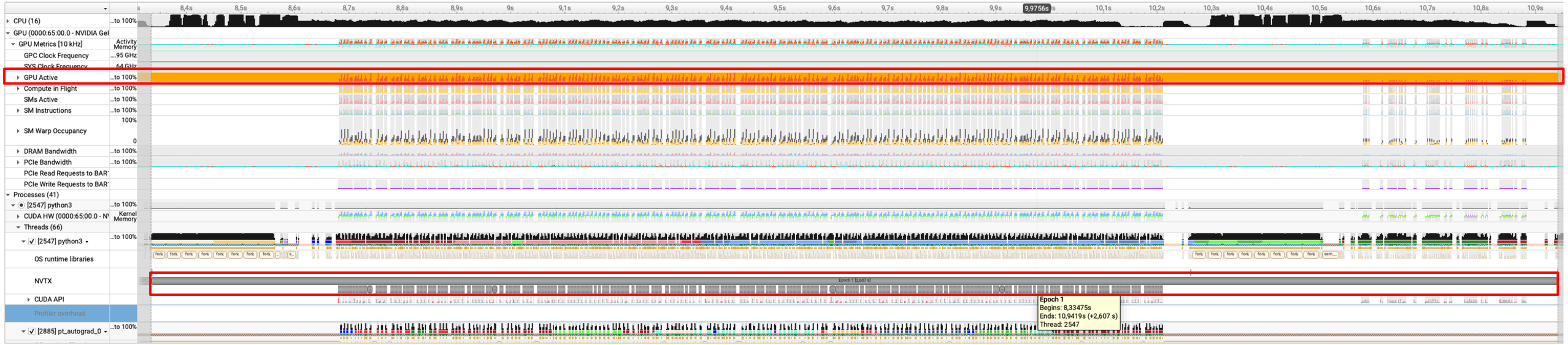
Nsight Systems profile for num_workers=2



- GPU a little more active, still mostly inactive
- Computation sparsity becoming better
- Runtime of about 4.9s per epoch

Optimizing a PyTorch Workload

Nsight Systems profile for num_workers=8



- Way better usage of GPU
- Only few “computation gaps” left
- Runtime of about 2.6s per epoch

Optimizing a PyTorch Workload

Timings for different number of workers

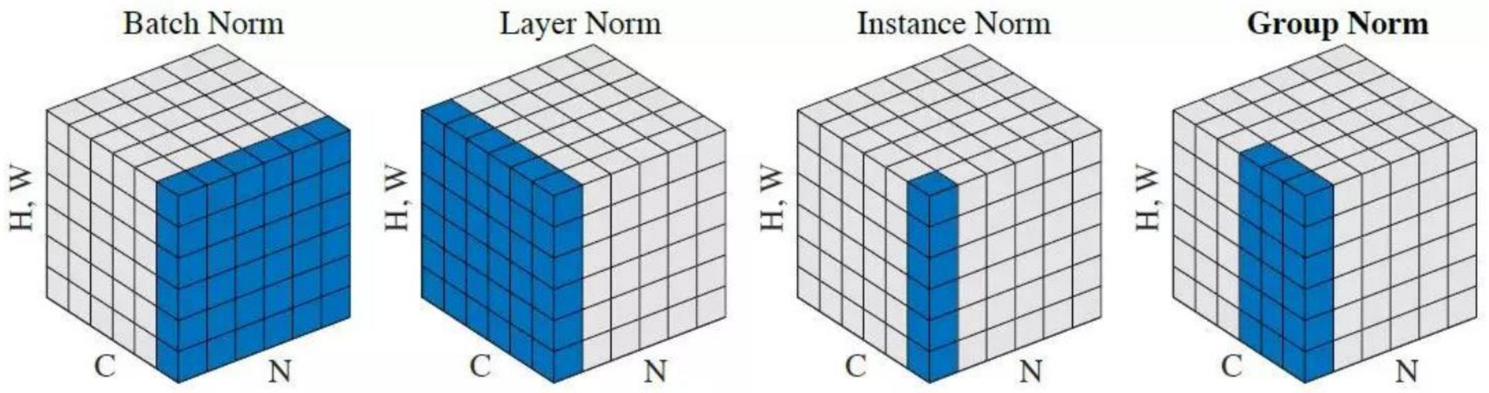
	Epoch Runtimes [s]	Relative speedup	Efficiency
num_workers=0	8.8	1.0	1.0
num_workers=2	4.9	1.8	0.9
num_workers=4	3.0	2.9	0.725
num_workers=8	2.6	3.4	0.425

- Can we do this indefinitely? No, because…
  - Limited by number of threads/cores
  - Limited by “computation gaps”
  - Limited by memory:

```
RuntimeError: DataLoader worker (pid 2123) is killed by signal: Bus error. It is possible that dataloader's workers are out of shared memory. Please try to raise your shared memory limit.
```

Instance Normalization

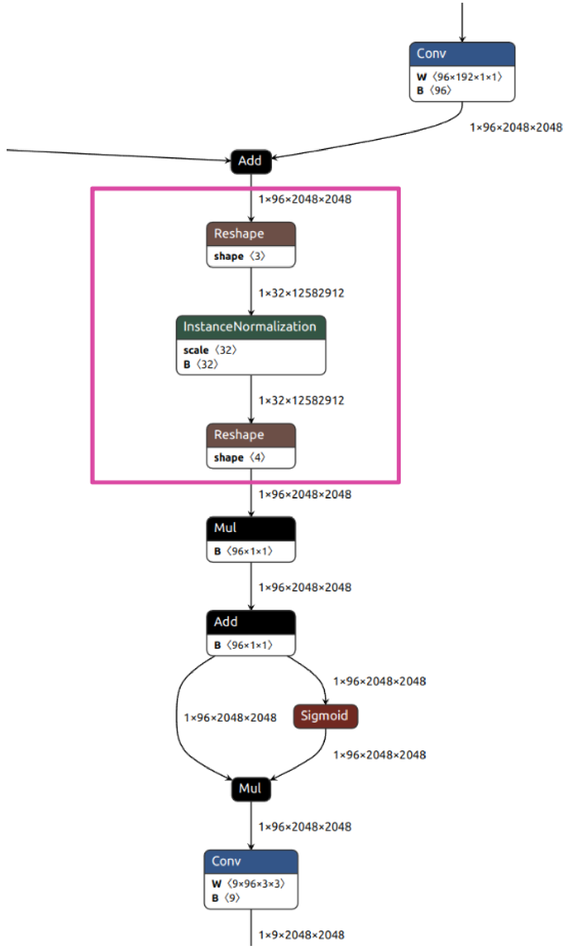
Mean and Variance based Normalization



Instance Normalization
Similar to other normalization methods,
but applied on a different "slice" of the tensor.

Super Resolution Network

Model Architecture



SR Architecture Block

Repeating blocks of Conv + InstanceNorm + Swish + Conv

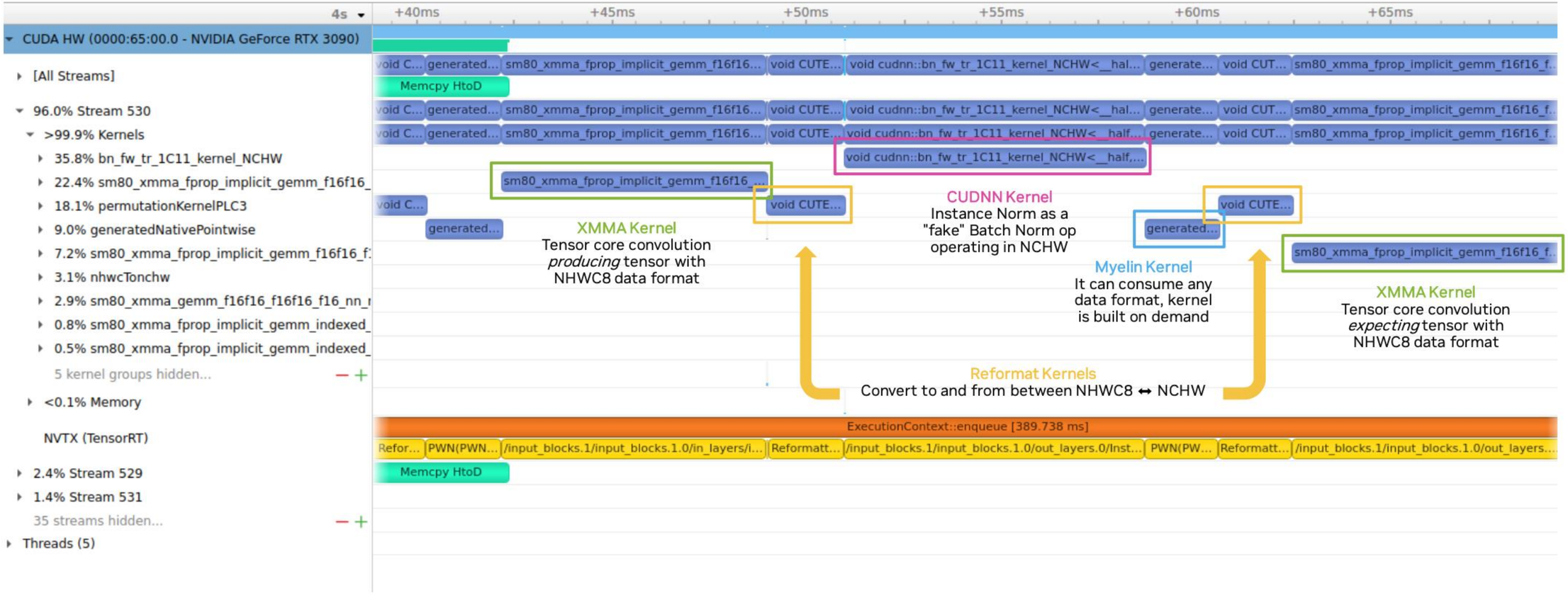
Instance Normalization

Initial Profile

One Iteration: 390 ms

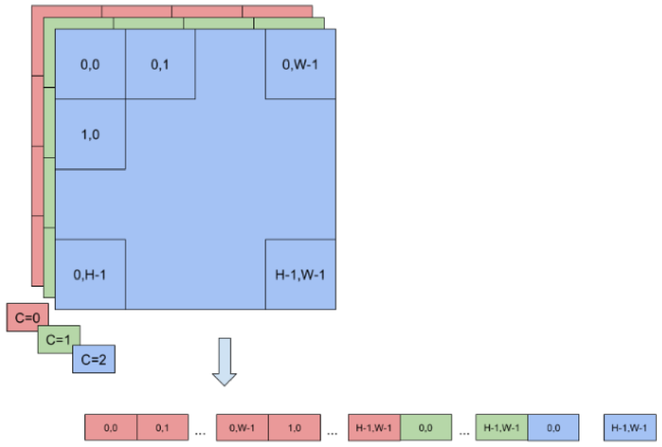
Instance Normalization

Zooming In: More Detail



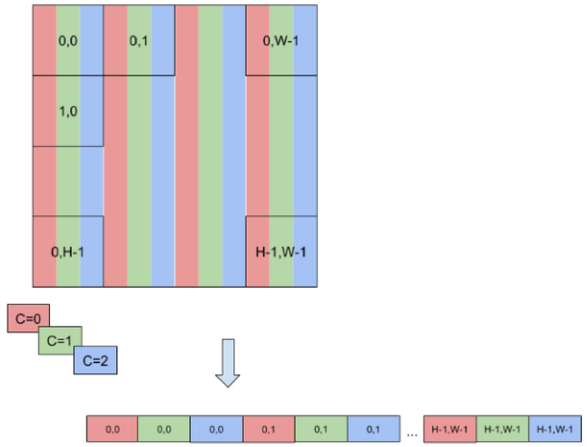
TensorRT Data Formats

Some of the most common tensor layouts

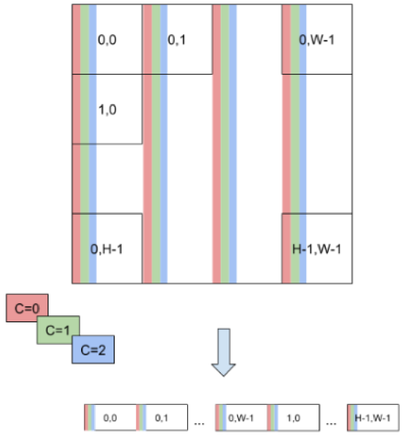


NCHW
Channel-first data format, spatial elements store linearly.

Standard for most TensorRT operations.



NHWC
Channel-last data format, elements interleaved by channel.

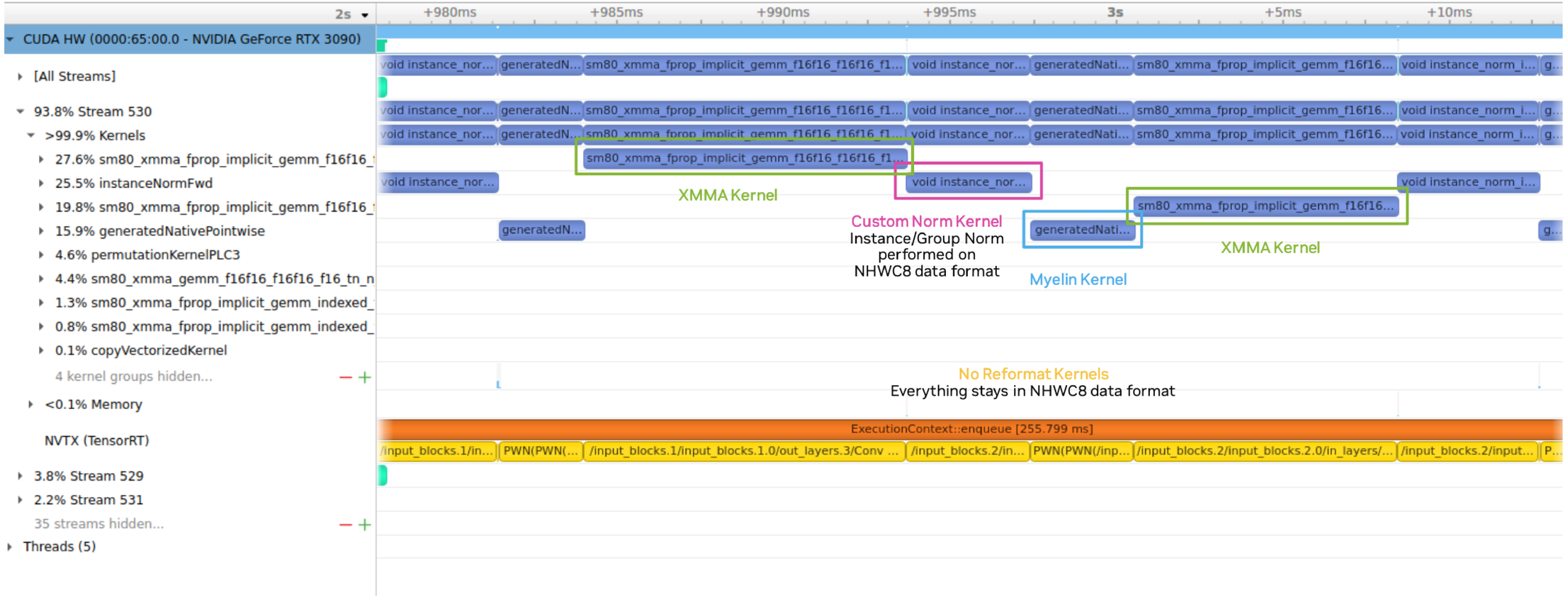


NHWC8
Channel-last data format, elements interleaved by channel. The channel dimension is aligned and padded to 8 (128-bit alignment).

Tensor Core FP16 specific.

Instance Normalization

Optimized NHWC8 Kernel



Instance Normalization

Optimized Profile

One Iteration: 256 ms

Instance Normalization

Results

Implementation	End-to-End Runtime	Instance Norm	Reformatting
Original Implementation	390 ms	139 ms	71 ms
NHWC8 Format Optimized	256 ms	65 ms	12 ms

RTX 3090, Batch Size 1, TensorRT FP16

1.5x Total Speedup

A better suited kernel for this type of data format and operation type

Reformatting largely reduced.

Nsight System Tricks

FAQ

- How to profile a multiple process application
  - One report for each process：horovodrun –np 8 nsys profile –o report_name_%q{HOROVOD_RANK} ./xxxxx
  - One report file for 8 processes：nsys profile –o report_name horovodrun –np 8 ./xxxx
  - Using conditional flag to capture only one process

```
if torch.distributed.get_rank() == 0 and condition1:
    torch.cuda.cudart().cudaProfilerStart()
if torch.distributed.get_rank() == 0 and condition2:
    torch.cuda.cudart().cudaProfilerStop()
```

+

```
nsys profile [nsys_args]
   --capture-range=cudaProfilerApi
   bash script.py [script_args]
```

- How to minimize your report file
  - Try to control report file size <100MB to speed up your analysis work.
  - using flags like“–y 60 –d 60 –s none”
  - using nsys launch + nsys start/stop style commands

Parameter	Default	Functionality	Example
-y, --delay	0	delay the collection after certain time (in seconds)	`nsys profile –y 60 ./appname # delay collection`
-d, --duration	N/A	duration of collection	`nsys profile –d 30 ./appname # only 30s are collected`
-s, --sample	process-tree	control the collection of CPU info	`nsys profile –s none #do not record CPU trace`

Nsight System Tips

FAQ

Continued…

- No GPU info contained in the file
  - Check if you have included the tracing flag
  - Adding nsys profile commands right before the python when you are using nested scripts.
  - using --trace-fork-before-exec flag

https://docs.nvidia.com/nsight-systems/UserGuide/index.html


NVIDIA