NVIDIA

Profiling Deep Learning with Nsight Systems

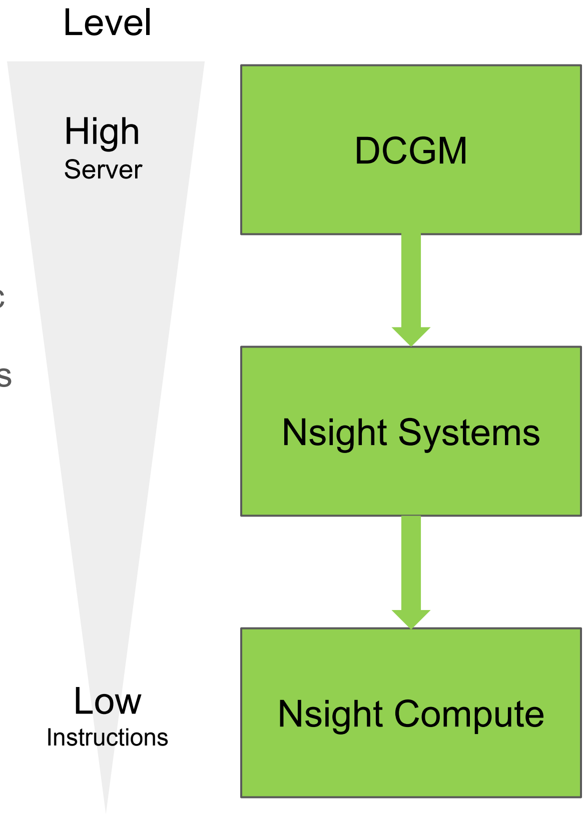
Daniel Horowitz
Director of Engineering
Platform Developer Tools

Agenda

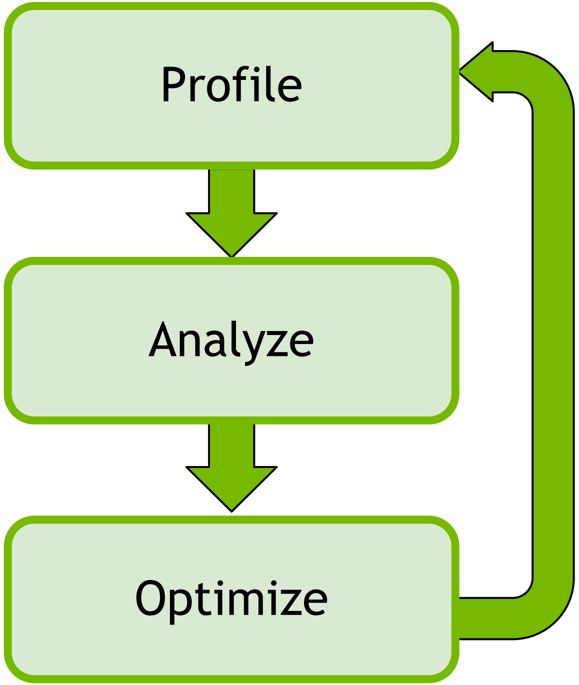
- Monitors vs Profilers
- Profiling on a single node
  - Overview
  - Basic features
  - GPU investigations
  - CPU investigations & features
  - Common bubble recovery tactics
- Profiling multi-node

Monitors vs Profilers

- Differ in the target users & intent = tradeoffs in design
  - Pick the right tool for the job!!!
  - Don’t expect one to be fully usable as the other
  - Gray area, hybrid amalgamations, but usually still tradeoffs
- Monitors
  - Goal: Coarsely observe quality, utilization, progress
  - Users: Machine and cluster admins, users of taskman, top, netstats, etc
  - Reports at a low rate (~10 - 1 hz)
    - Sensible to observer reacting, low overhead, smooth out numbers
    - Does not correlate back to code
  - Track issues back to jobs
    - Maybe minutes or seconds but often not root cause
  - Ex: DCGM
- Profilers
  - Goal: Aid program optimization
  - Users: Engineers looking to relating back to areas of code
  - Many different breeds of profiler!!!
    - Levels of observibility
    - Overhead trade-offs
  - Often not designed for 100% up-time and continual reporting like a monitor
  - Ex: Nsight Systems and Nsight Compute



Profiler-Driven Optimization Workflow



Iterate until desired performance is achieved

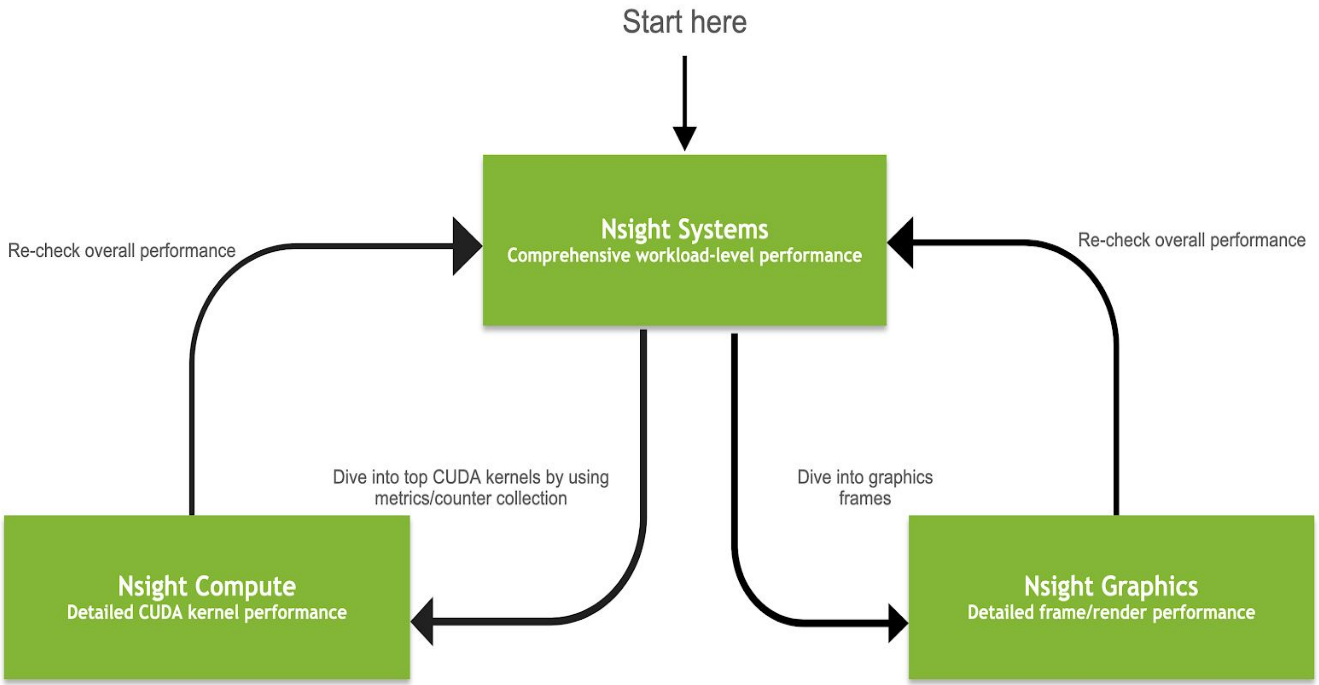
Nsight Profiler Family

Performance Analysis Workflow

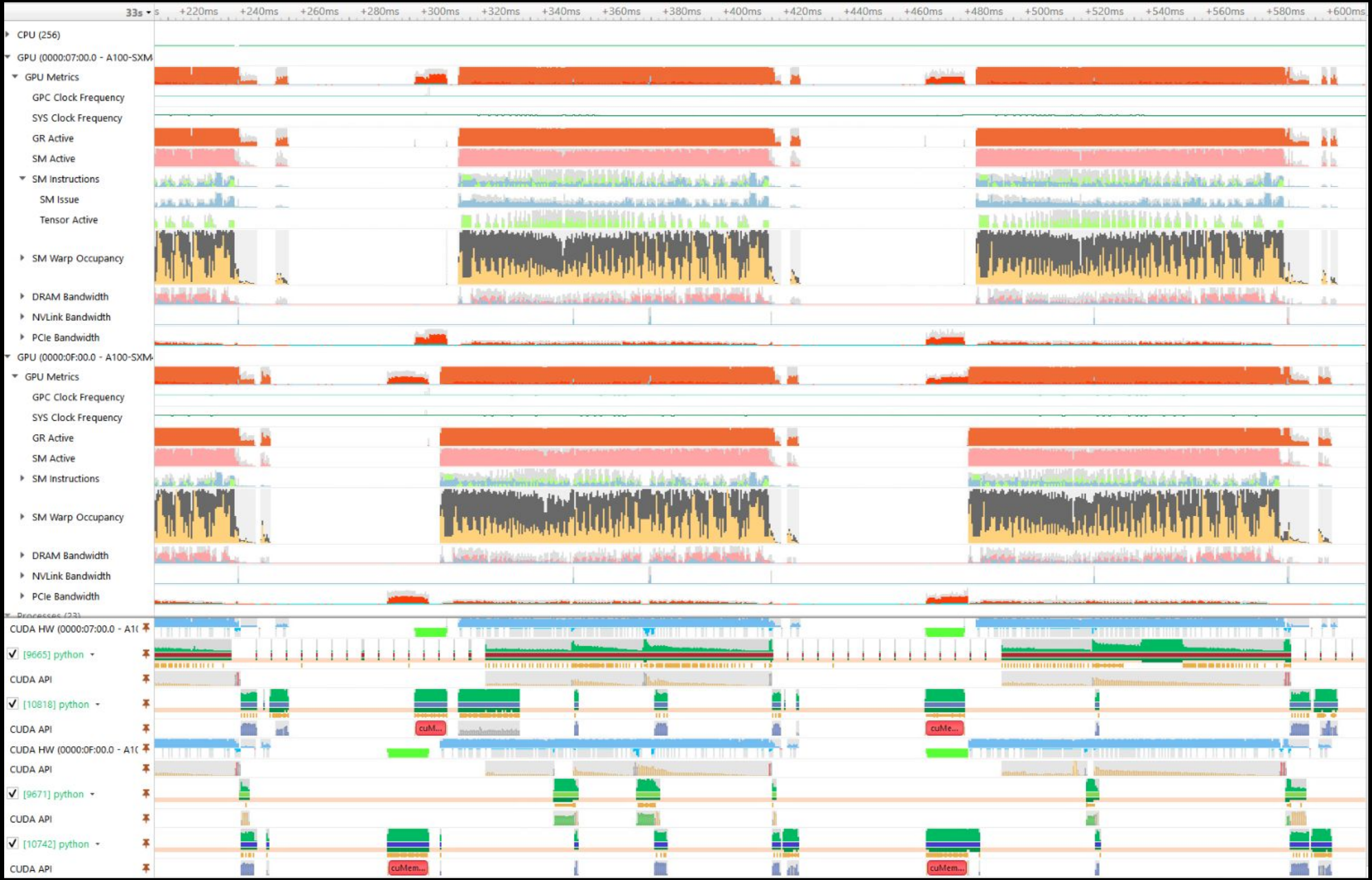
Nsight Systems -
Analyze application algorithm system-wide CPU,GPU,CUDA,Graphics

Nsight Compute -
Debug/Optimize CUDA kernels

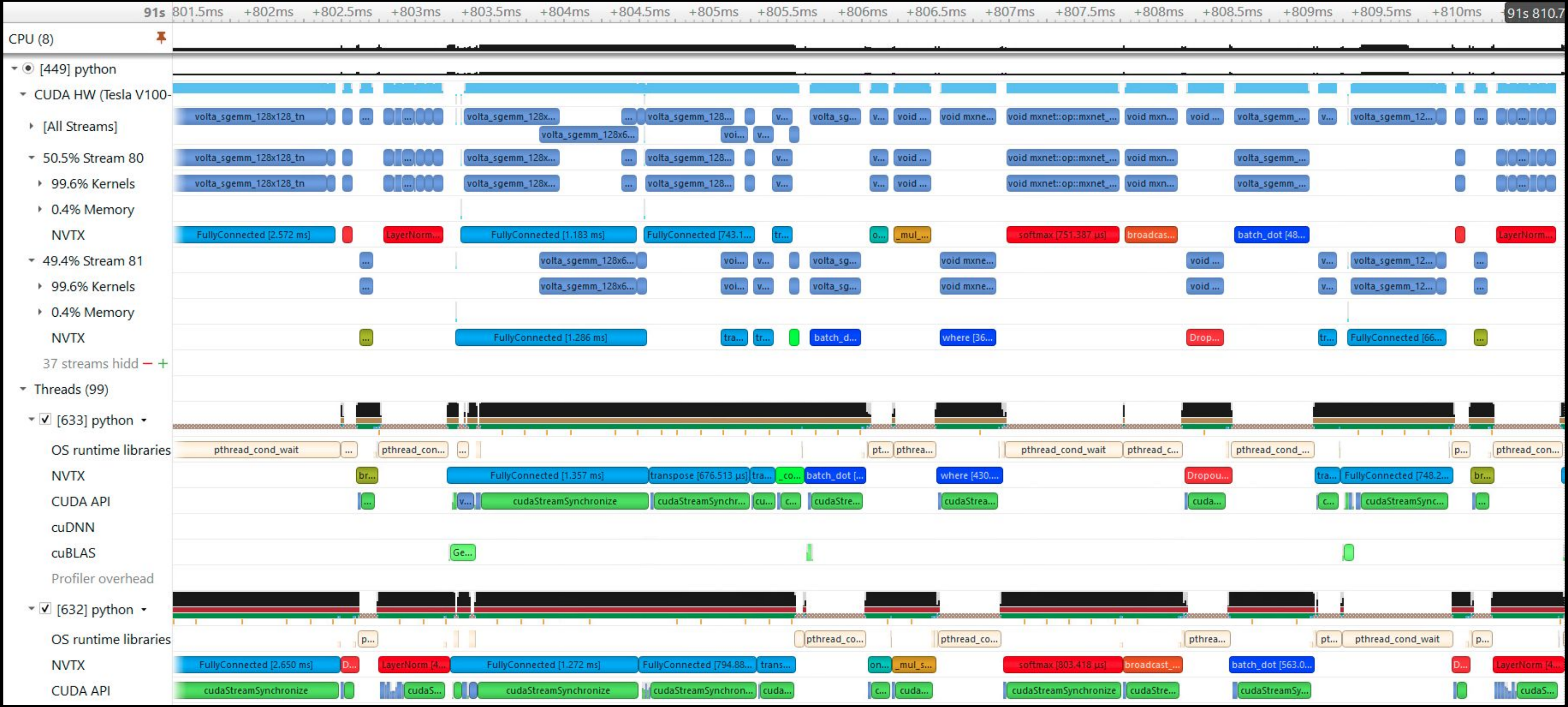
Nsight Graphics -
Debug/Optimize Graphics shaders & frames



Nsight Systems

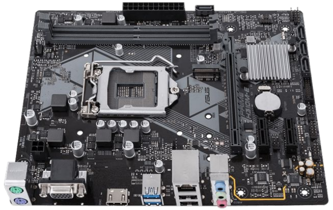


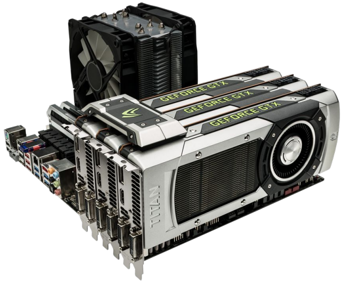
Timeline looks like a spectrogram when zoomed out
Ex: DL model training, 2 iterations, GPU & CPU



Zooming in transforms many row graphs into ranges

Tuning an Orchestra of Tasks





Why start with Nsight Systems?

It’s designed to allow you to…

See the big picture

See how asynchronous CPUs, GPUs, NICs, and software are interacting

Who is stuck on whom

Measure the higher-level costs to pick the best opportunities

Avoid work based on intuition & false-positive indicators

- Don’t assume GPU bound and skip ahead to other tools
- Some synchronization oriented issues can look like GPU bound

Statistics alone often aren’t enough info to understand and resolve the issue

WARNING: The presenter & slides will often refer to “Nsight Systems” as Nsys or NSYS

Overview

- System-wide application algorithm tuning
- Locate optimization opportunities
  - Visualize millions of events on a very fast GUI timeline
  - See gaps of unused CPU and GPU time
- Balance your workload across multiple GPUs, CPUs, NICs
  - GPU streams, kernels, memory transfers, etc
  - CPU algorithms, utilization, and thread state
- UX: CLI, GUI timeline, statistics, data export
  - Ex: Collect nsys-rep w/CLI on cluster, SCP to PC to view, mine either place
- Multi-platform: Linux & Windows, x86-64, ARM server, Tegra
  - Mac (host-only)

Timeline Features

GPU

- GPU HW metrics sampling
- CUDA GPU-side kernel and mem-op ranges correlated to CPU API calls
- Libraries: cuBLAS, cuDNN, cuDF, TensorRT

CPU, OS, & application

- Thread state, migrations, and call-stacks
- OS runtime long call trace (>1us, pthread, glibc → mmap, file & IO, ...)
  - Call-stack backtraces (>80us)
- ftrace or WDDM & ETW (page faults, signal, interrupts, …)

Code Annotations APIs

- NVTX

Networking

- UCX, MPI (OpenMPI & MPICH), OpenSHMEM, NVSHMEM, NCCL trace
- NVIDIA NIC/HCA metrics sampling (Infiniband & Ethernet)

Wait!!! I just want stats about my DNN ☹️

You have barely mentioned Deep Learning!!!

Each DL framework has its own tools, so we'll discuss how to go deeper

Many DL frameworks have their DNN layer execution instrumented with NVTX

- Sometimes just the NVIDIA container if the framework doesn't accept it upstream

Nsys has the relationships: DNN layer → CUDA Launch → GPU CUDA Kernel Execution

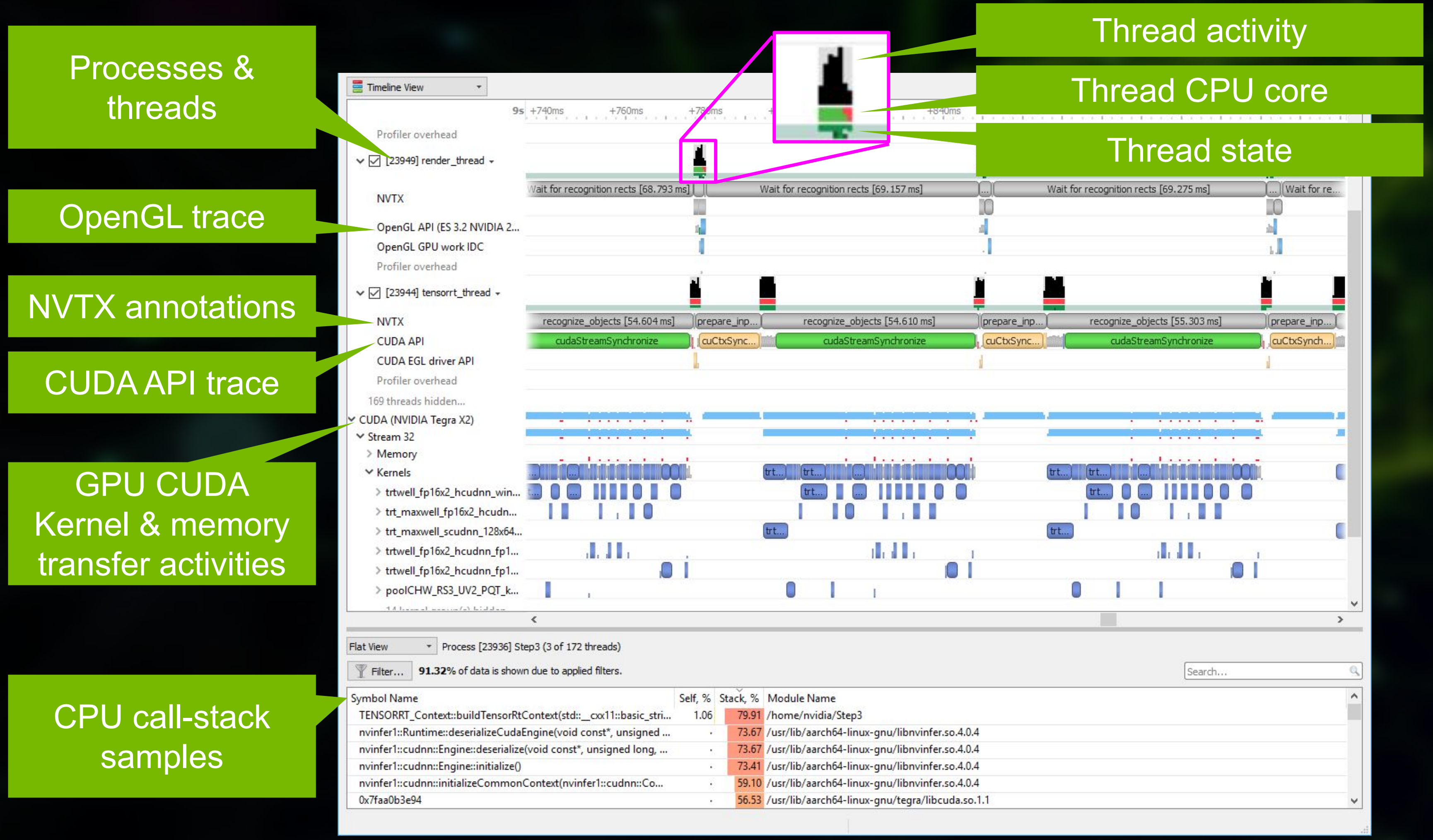
Nsys can export to SQLite

Nsys has python scripts & documentation on how to analyze that database

If that's all you want, that's the easy path

But we're about to go deeper and show you how it's executing!!!

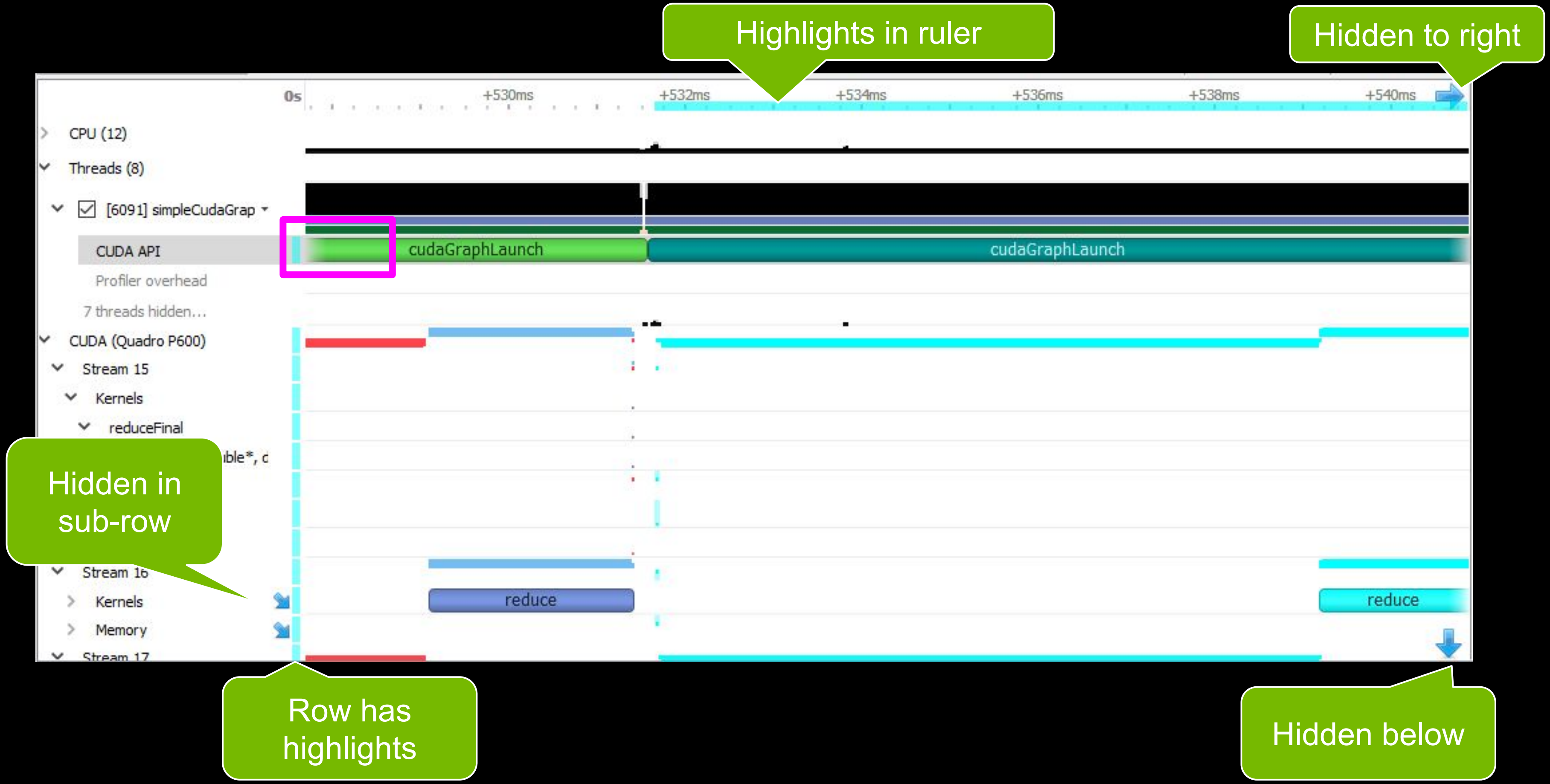
So you can visually investigate, craft better stats, and create your own expert systems



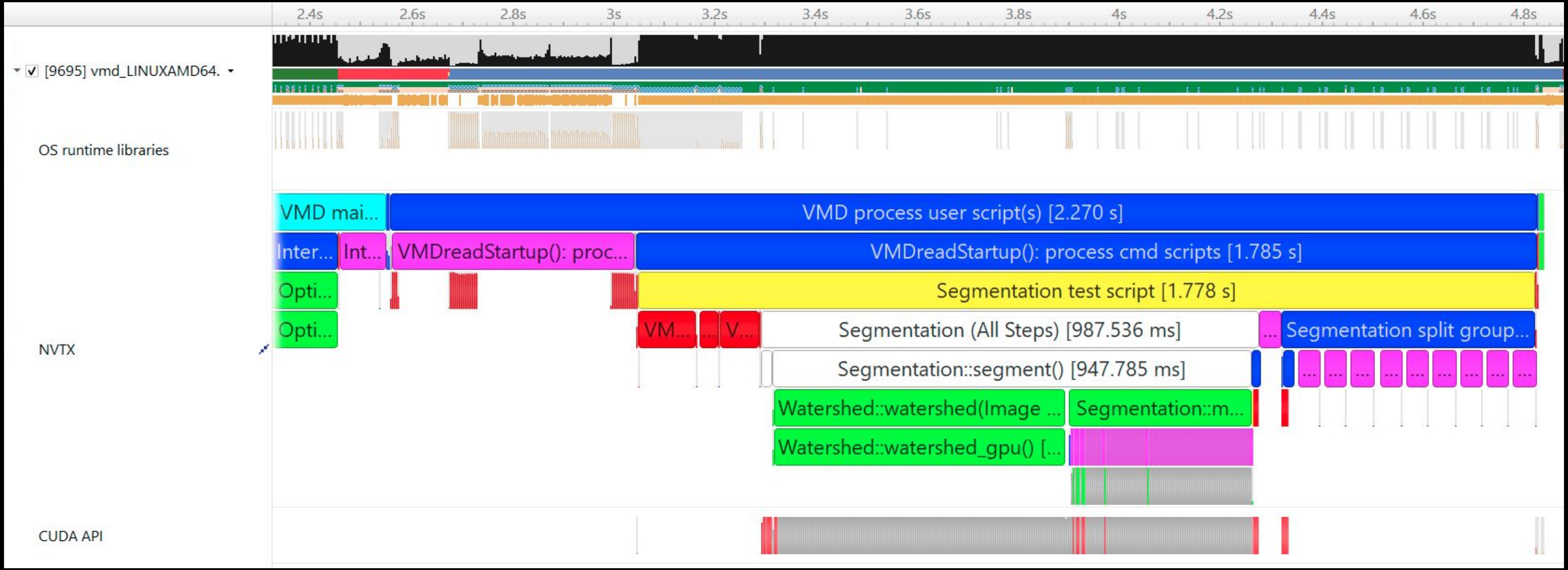
Processes & threads
OpenGL trace
NVTX annotations
CUDA API trace
GPU CUDA Kernel & memory transfer activities
CPU call-stack samples
Thread activity
Thread CPU core
Thread state
Timeline View
Profiler overhead
[23949] render_thread
NVTX
OpenGL API (ES 3.2 NVIDIA 2...
OpenGL GPU work IDC
[23944] tensorrt_thread
CUDA API
CUDA EGL driver API
169 threads hidden...
CUDA (NVIDIA Tegra X2)
Stream 32
Memory
Kernels
trtwell_fp16x2_hcudnn_win...
trt_maxwell_fp16x2_hcudn...
trt_maxwell_scudnn_128x64...
trtwell_fp16x2_hcudnn_fp1...
poolCHW_RS3_UV2_PQT_k...
Wait for recognition rects [68.793 ms]
Wait for recognition rects [69.157 ms]
Wait for recognition rects [69.275 ms]
recognize_objects [54.604 ms]
recognize_objects [54.610 ms]
recognize_objects [55.303 ms]
prepare_inp...
cudaStreamSynchronize
cuCtxSync...
Flat View
Process [23936] Step3 (3 of 172 threads)
Filter...
91.32% of data is shown due to applied filters.
Search...
Symbol Name
Self, %
Stack, %
Module Name
TENSORRT_Context::buildTensorRtContext(std::__cxx11::basic_stri... 1.06 79.91 /home/nvidia/Step3
nvinfer1::Runtime::deserializeCudaEngine(void const*, unsigned ... 73.67 /usr/lib/aarch64-linux-gnu/libnvinfer.so.4.0.4
nvinfer1::cudnn::Engine::deserialize(void const*, unsigned long, ... 73.67 /usr/lib/aarch64-linux-gnu/libnvinfer.so.4.0.4
nvinfer1::cudnn::Engine::initialize() 73.41 /usr/lib/aarch64-linux-gnu/libnvinfer.so.4.0.4
nvinfer1::cudnn::initializeCommonContext(nvinfer1::cudnn::Co... 59.10 /usr/lib/aarch64-linux-gnu/libnvinfer.so.4.0.4
0x7faa0b3e94 56.53 /usr/lib/aarch64-linux-gnu/tegra/libcuda.so.1.1

CUDA

API launch to GPU kernel correlation

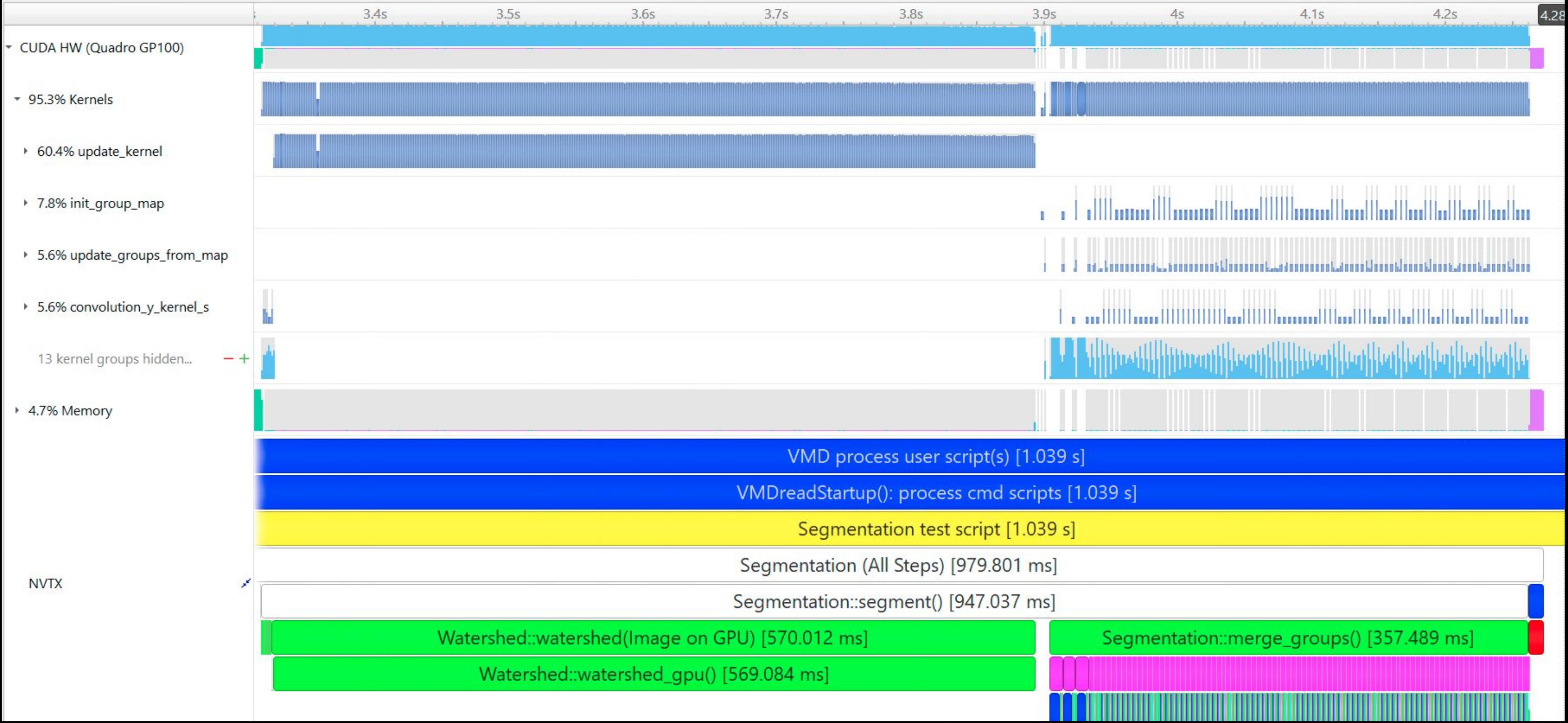


CPU-GPU correlation & location assistance



Code annotations APIs
NVTX = NVIDIA Tools eXtensions

Example: Visual Molecular Dynamics (VMD) algorithms visualized with NVTX on CPU



NVTX ranges project from CPU onto the GPU CUDA streams

Example: Visual Molecular Dynamics (VMD) algorithms visualized with NVTX on GPU

PyTorch

DNN layer annotations are disabled by default

Add the following: “with torch.autograd.profiler.emit_nvtx():”

TensorRT is also annotated already if that is the backend you are using

You can also add NVTX to your python script manually:

https://github.com/pytorch/pytorch/blob/master/torch/cuda/nvtx.py
https://pytorch.org/docs/stable/generated/torch.cuda.nvtx.range_push.html
https://pytorch.org/docs/stable/generated/torch.cuda.nvtx.range_pop.html

TensorFlow

DNN layers are annotated **by default** with NVTX in NVIDIA TF containers!

https://catalog.ngc.nvidia.com/orgs/nvidia/containers/tensorflow

https://github.com/NVIDIA/tensorflow

TF_DISABLE_NVTX_RANGES=1 if you want to disable for production

Adding more detail to the timeline for setup,eager mode, tf.data.Dataset.from_generator, etc

https://github.com/NVIDIA/NVTX/tree/dev/python

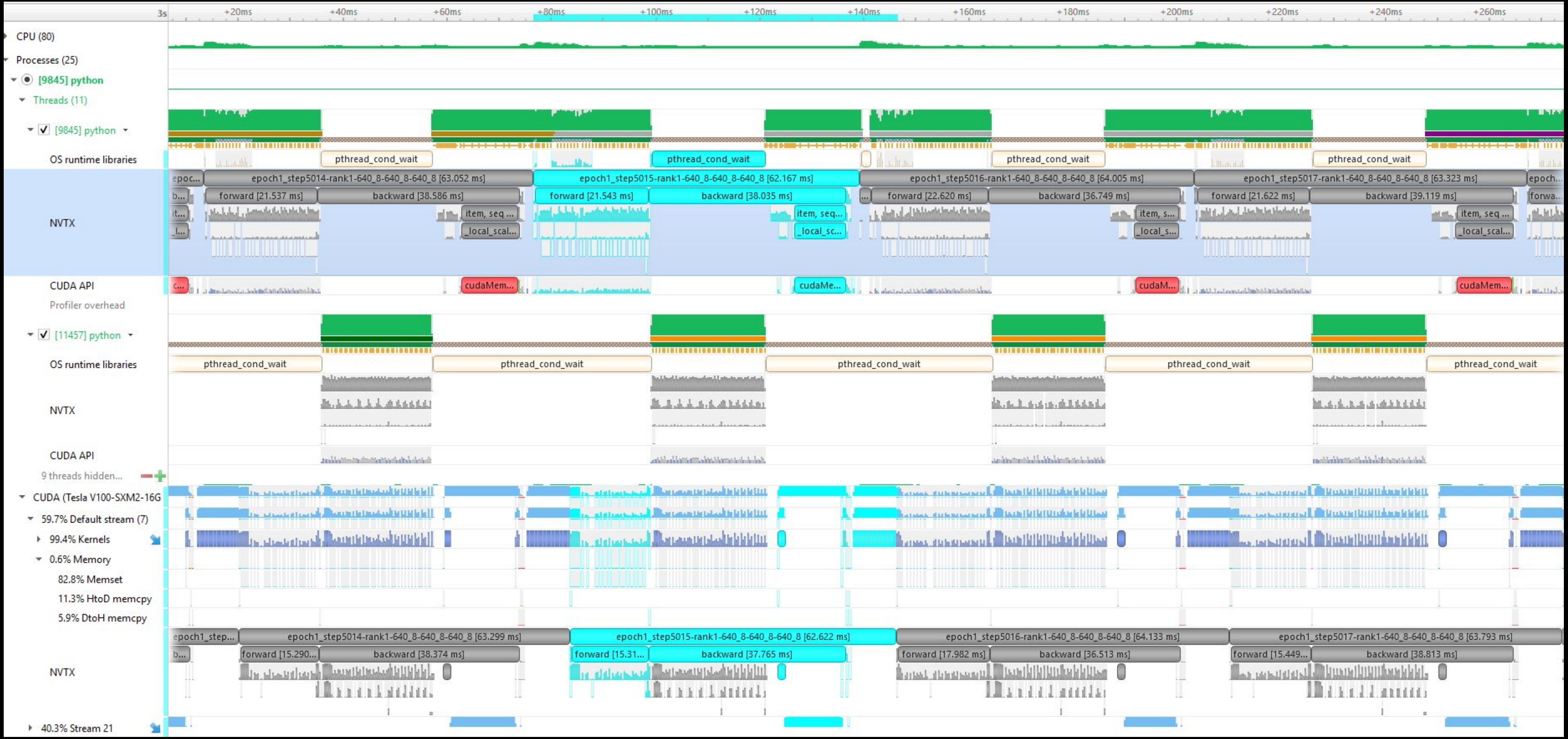
```
nsys profile --trace=cuda,nvtx,osrt,cudnn,cublas --backtrace=dwarf
--capture-range=cudaProfilerApi --gpu-metrics-devices=all --output=oft-profile-dwarf4
sh scripts/expt-singleGPU.sh --profile 50 --profile_start 5000 --profile_epoch 1
```

Nsight Systems CLI command
Select APIs to trace
Enable GPU memory use tracking (but there is extra overhead)
Collect thread call-stack sample backtraces via DWARF info - deeper but more expensive to collect
Trigger collection on cudaProfilerStart API in application, or consider timer-based options
GPU metrics sampling at default 10khz
Name the report file
Application command - plus arguments for when to start profiling

https://docs.nvidia.com/nsight-systems/UserGuide/index.html#cli-profile-command-switch-options

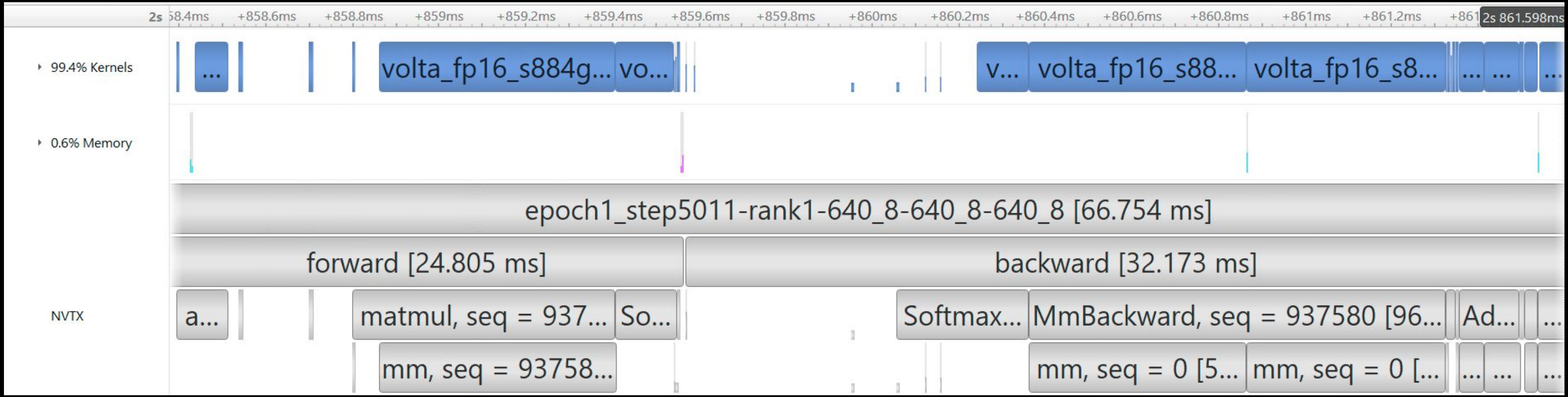
srun nsys profile … required on multi-node or multi-container
nsys profile mpirun … optional on single node to produce a single report

Nsight Systems CLI command line example



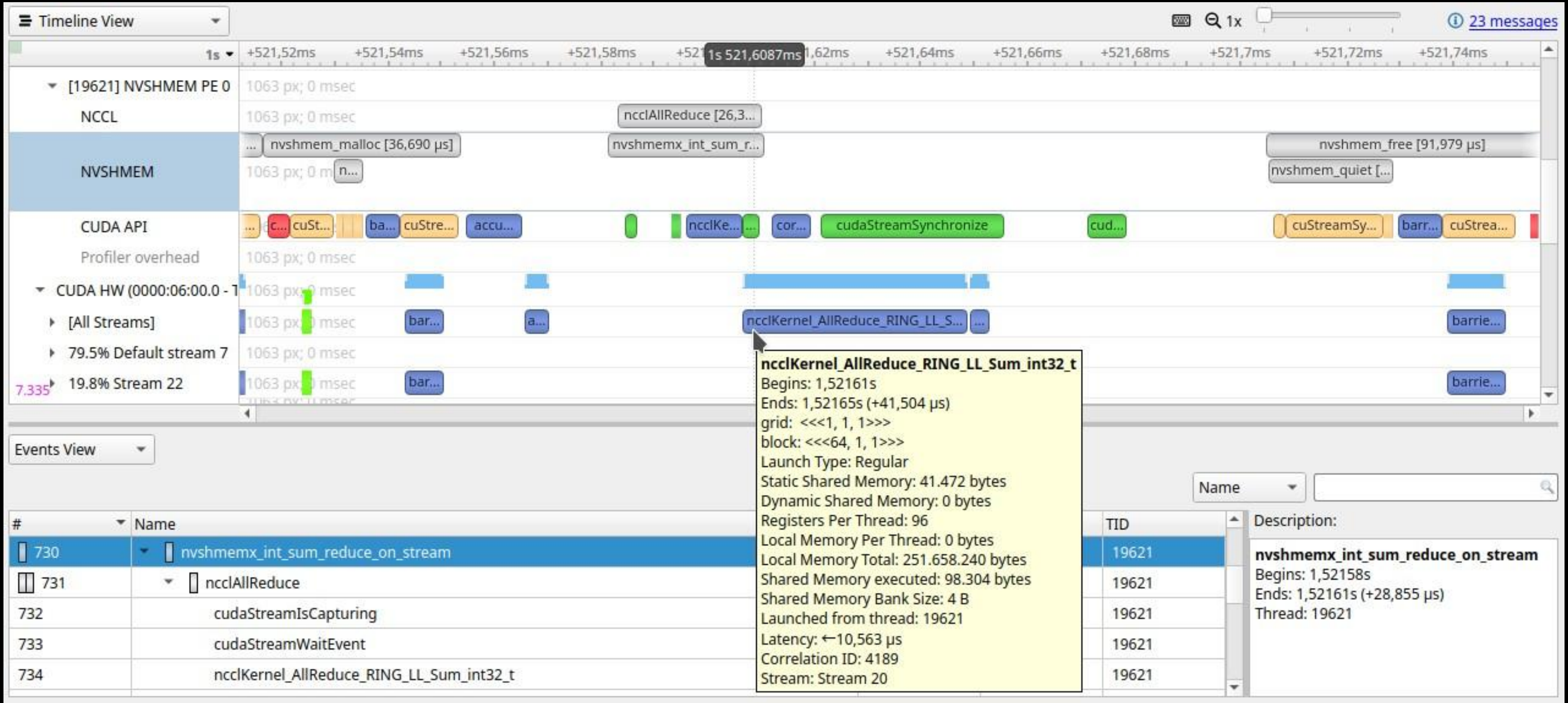
PyTorch Transformer with NVTX ranges projected onto the GPU

PyTorch Transformer with NVTX ranges projected onto the GPU

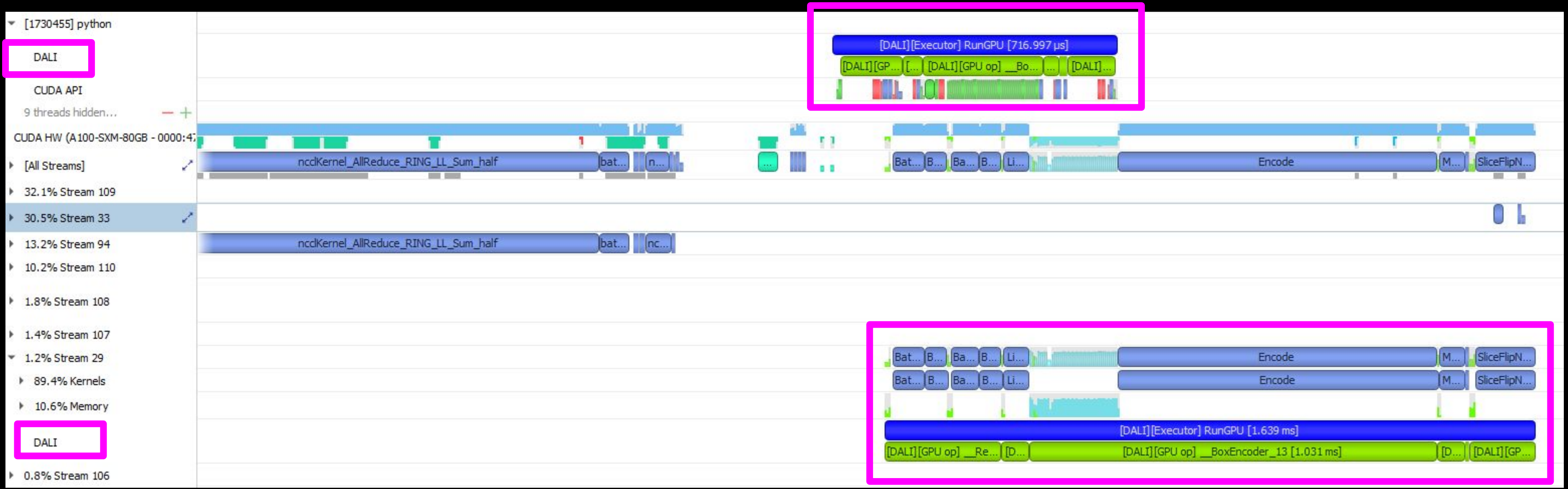


Example: NVTX on GPU in PyTorch transformer model in eager mode (ie non-hybrid)

TensorFlow Resnet50 DNN nodes as NVTX ranges projected onto the GPU



NVSHMEM & NCCL



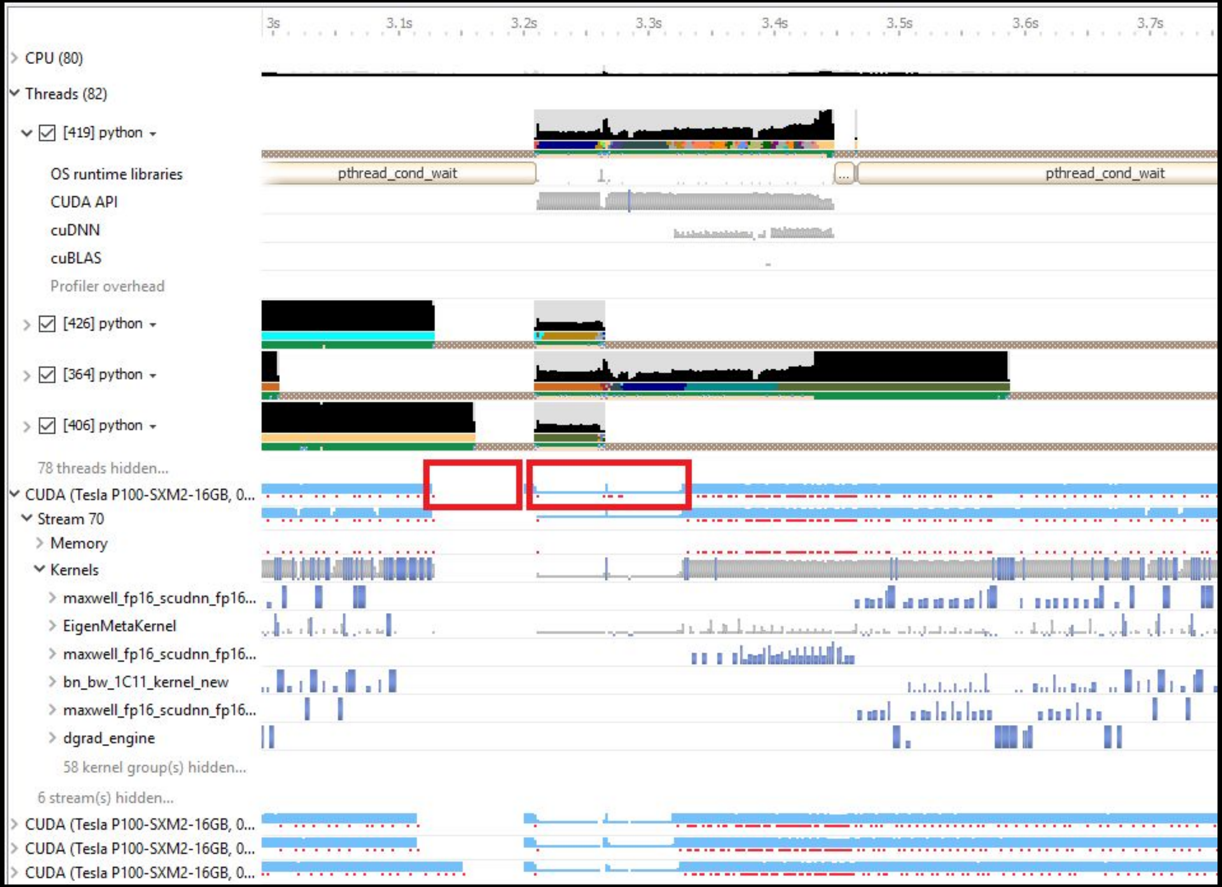
NVIDIA DALI trace
Data Loading Library

Core Investigation Strategy

- What’s HOT?
  - Will it be easier to shrink what i have?
  - This is where MOST people concentrate. …intuitive but not always better
- **What’s COLD?**
  - Will it be easier to take advantage of the something unused?
  - **Free money? Yes please!**
- Is it doing what I intended & budgets? (hint: often not as well as you thought)
- Cold spots are often clear, measurable opportunities!!!
  - How can i remove or fill them?
  - Where do i have incorrect/unnecessary/unexpected dependencies/synchronization?
- Hot spots
  - Might be parallelizable?
  - Might not be shrinkable without compromising accuracy, memory, etc

GPU Bubble Detective

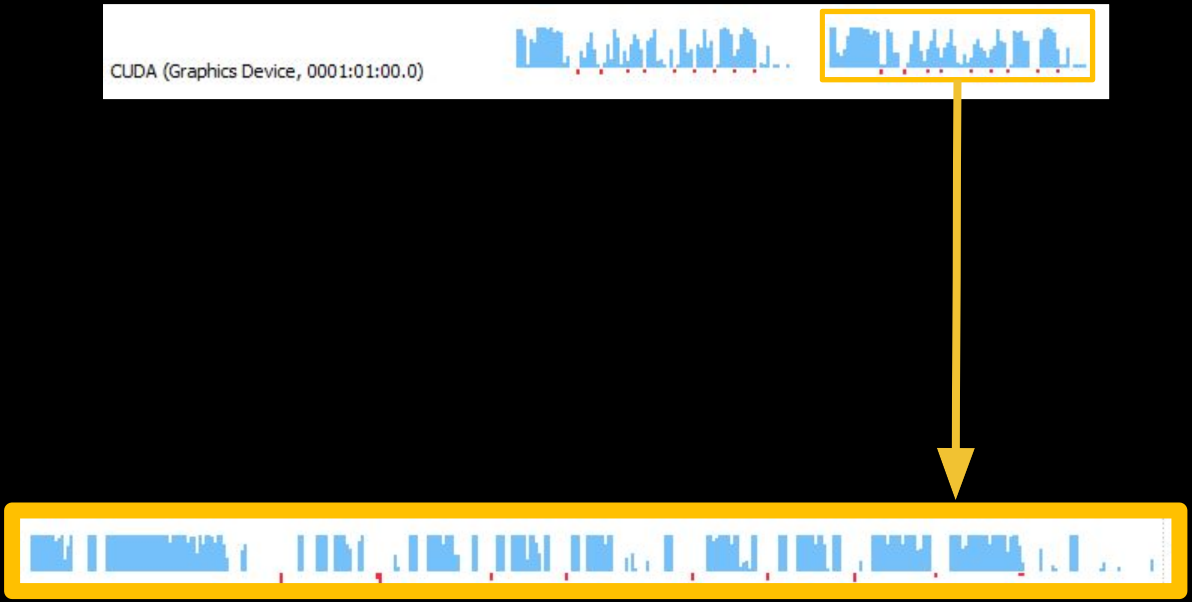
- Find the crime (cold or cool spot)
- Use correlation to track back to the CPU
  - Select surrounding GPU CUDA ops
- Investigate what was in the gap
  - Thread call-stack backtrace samples
  - OS Runtime long function backtraces
  - API & library traces
  - User-coded annotations



CUDA trace based GPU idle and low utilization level of detail

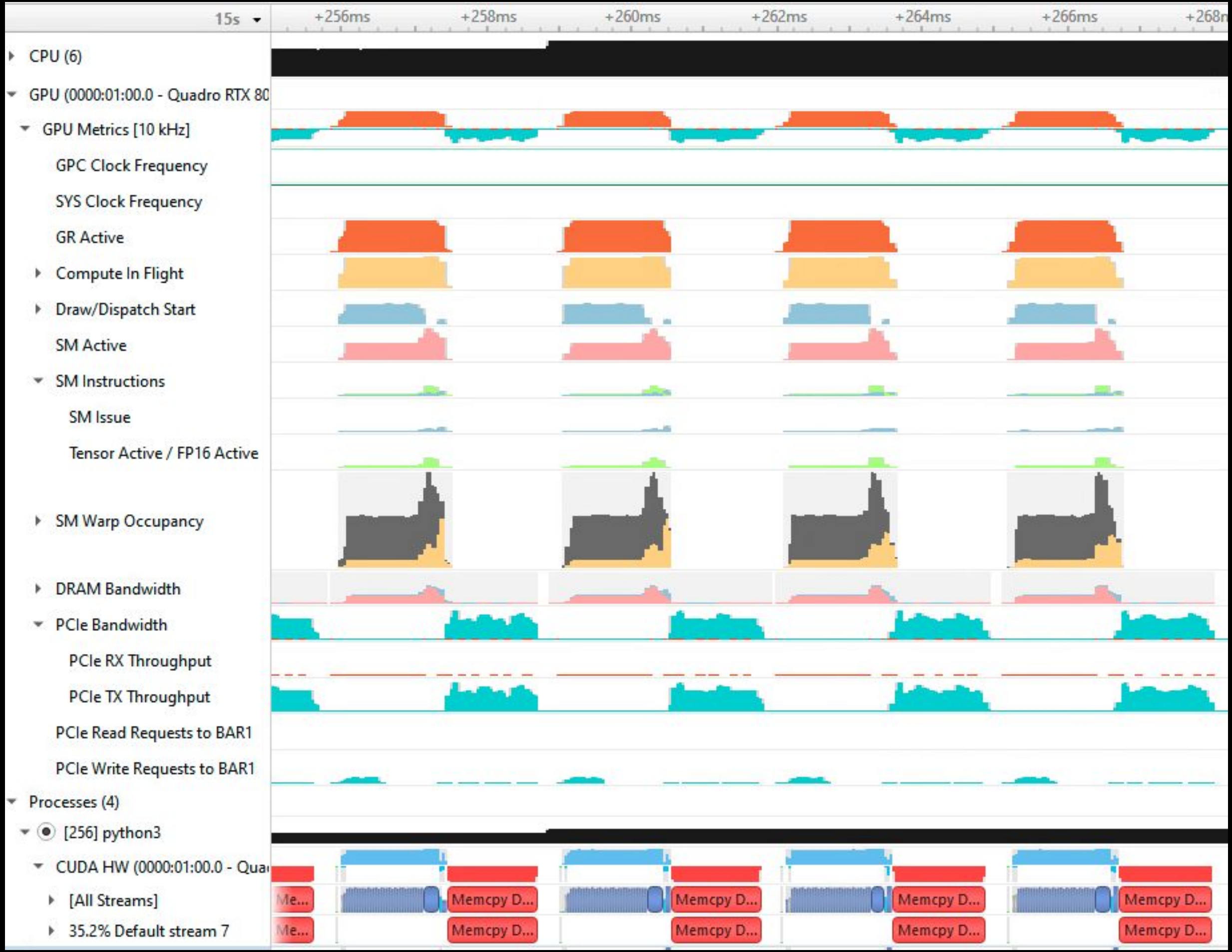
Investigate by select kernels around bubble to find related CPU range

CUDA GPU utilization graph is based on percentage time coverage



Zooming in reveals gaps where there were valleys

CUDA streams eventually convert from graphs to ranges



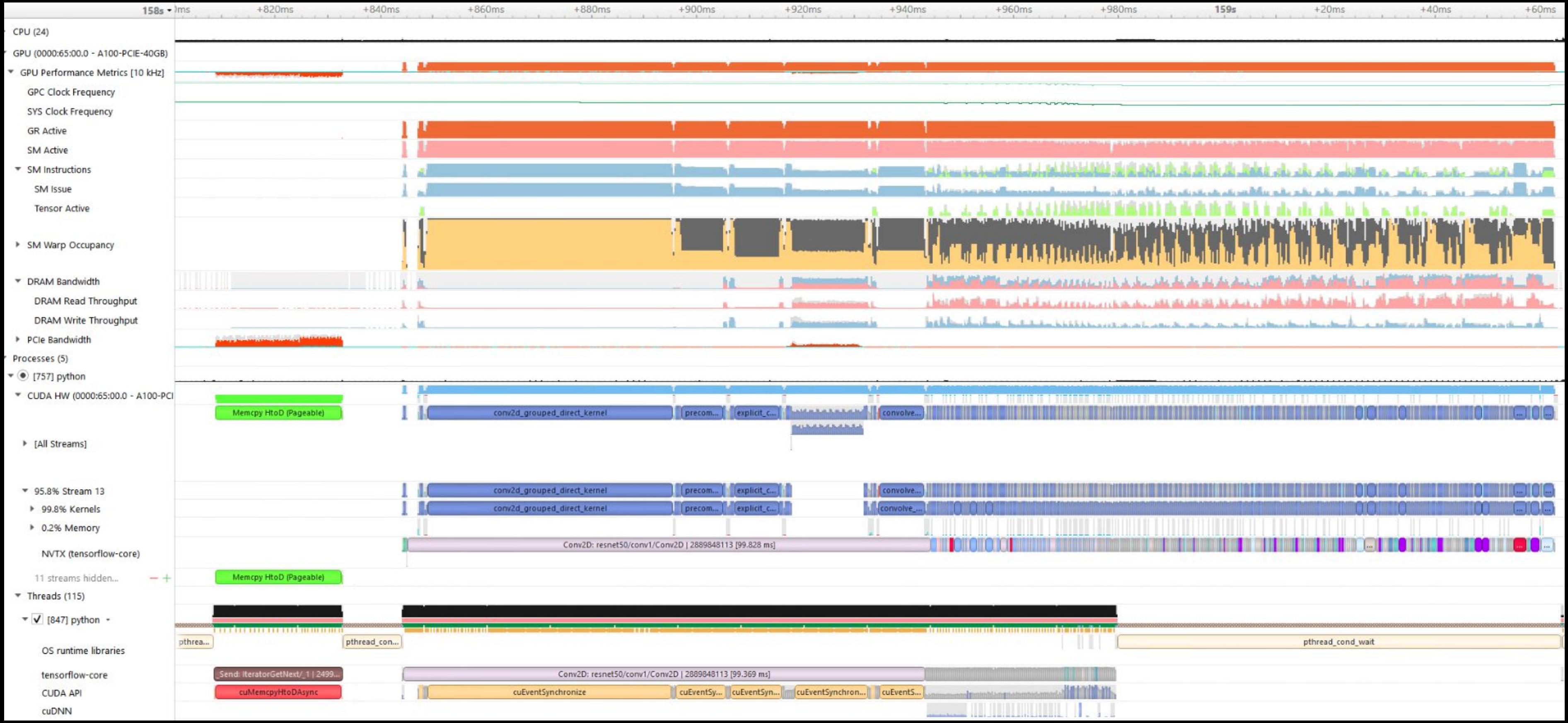
GPU Metrics Sampling - how to interpret it

Interpreting GPU Metrics Sampling

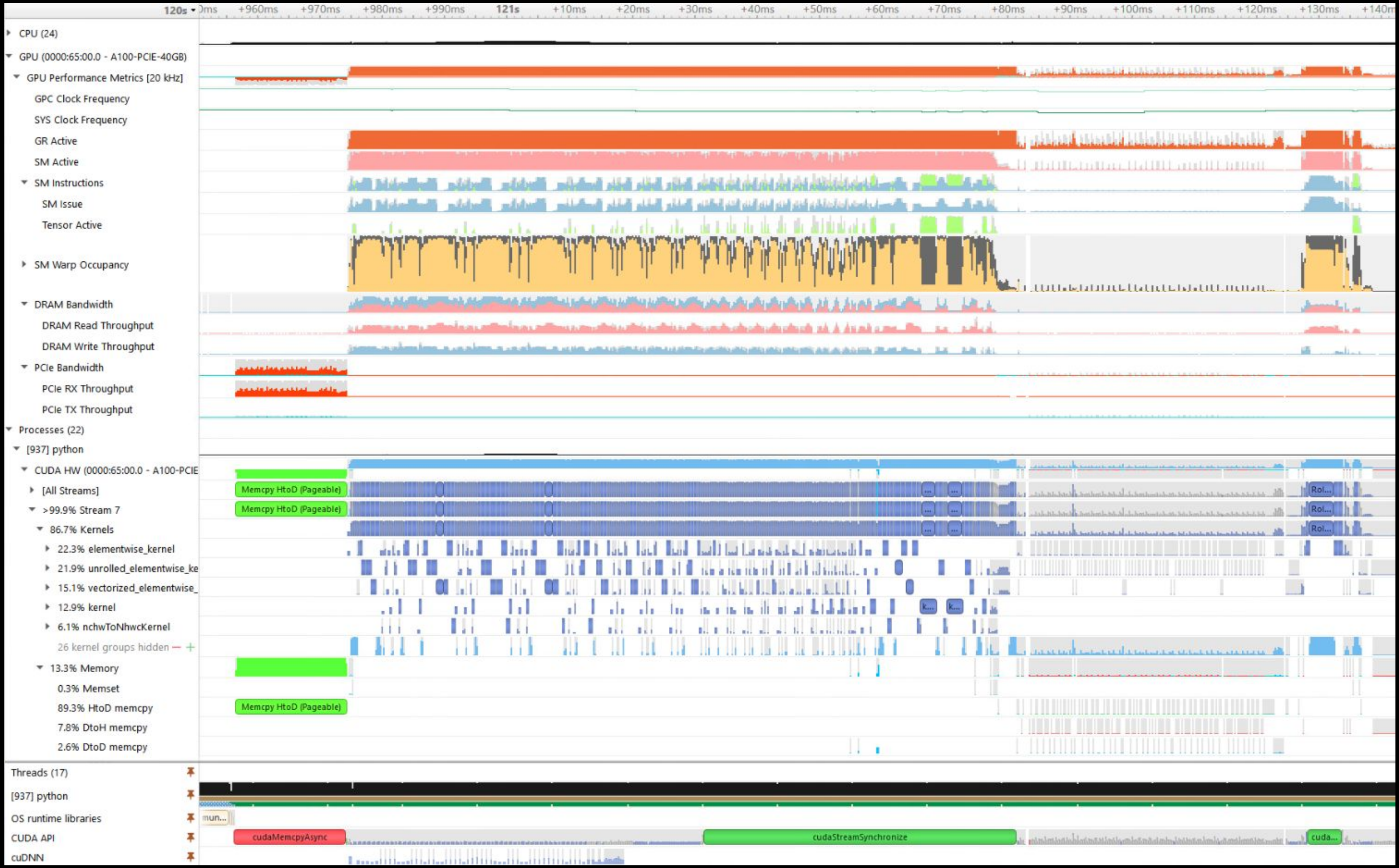
- More info, no trace overhead, collected device-wide OOP
  - But no kernel names

- GR Active ⇒ It’s doing some work
  - % GPU has any SM active (or NVENC, NVDEC, graphics)
- SM Active ⇒ How well is it using the width of the GPU
  - If low, increase batch sizes or look at kernel grid dimension
- SM Instructions Issued ⇒ Is it performing a lot of instructions
  - or might I be waiting on memory if low
  - Not enough warps to cover memory latency; larger kernel block dimensions can help
- SM Instructions tensor active ⇒ Using very faster special hardware
  - performance gains but slightly counter SM instructions can drop (vary by architecture)
  - can be limited by SM shared memory & waiting for loads
- Warp occupancy ⇒ Ratio of SM code types
  - Don’t optimize for this! Ultra-optimized kernels don’t always maximize warps!!! Ex: cuBLAS
- Memory and bus activity

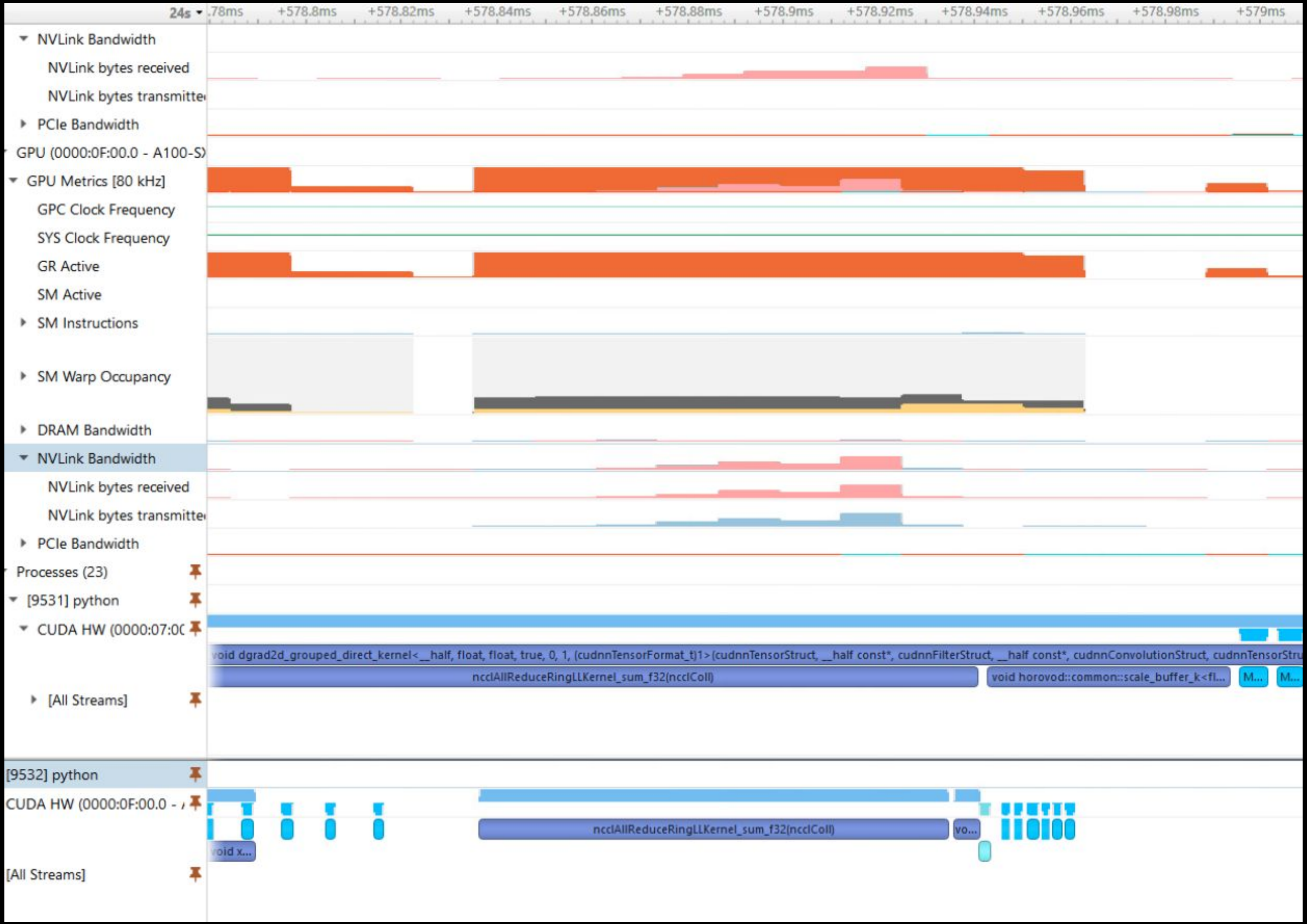
- NOTE 1: Requires disabling DCGM and any DL framework built-in profiler
- NOTE 2: For lower priv access https://developer.nvidia.com/ERR_NVGPUCTRPERM



GPU Metrics Sampling
Ex: TensorFlow2 ResNet50



GPU Metrics Sampling – Mask-RCNN



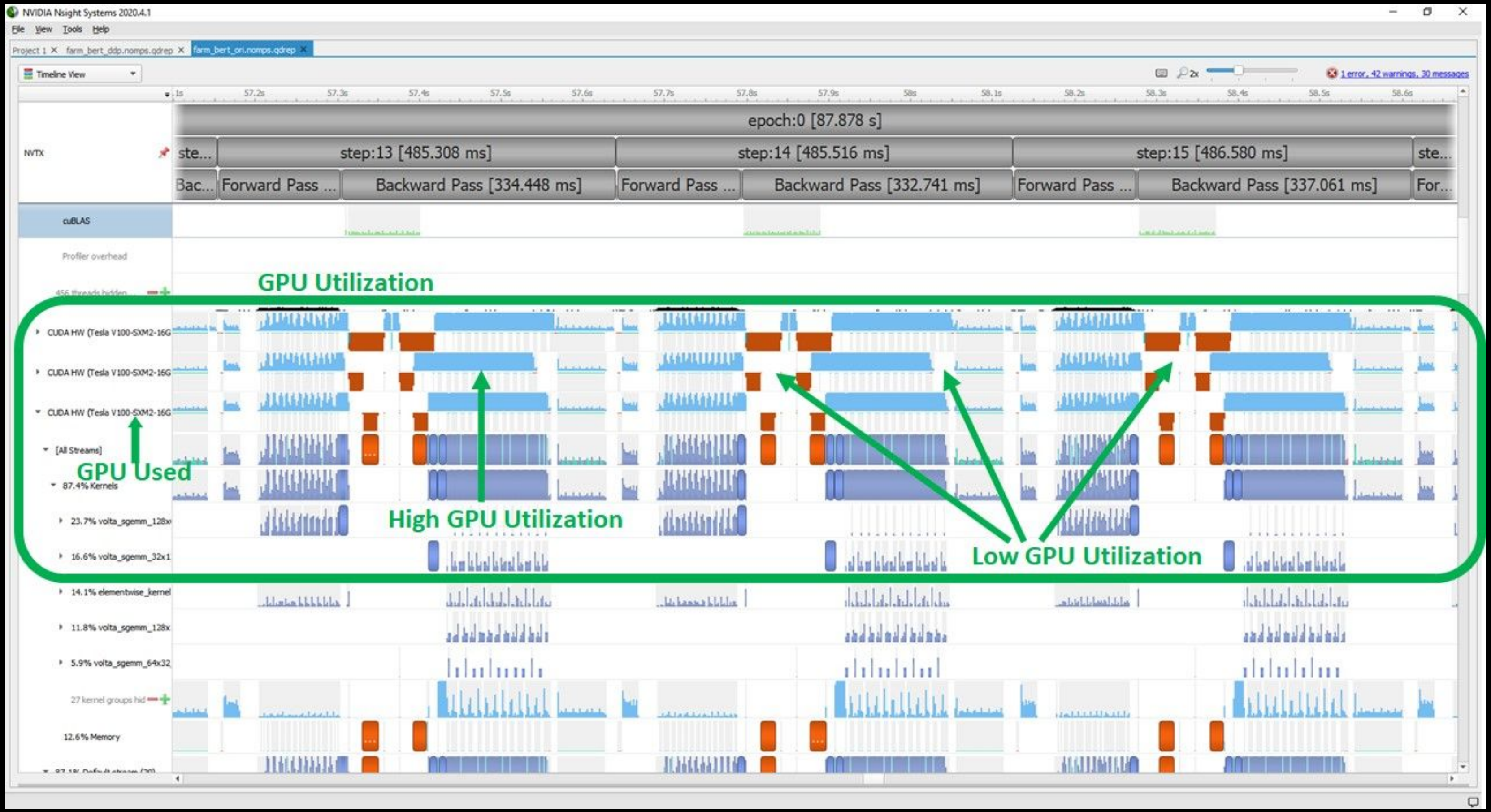
GPU Metrics Sampling - NCCL using NVLink

Optimization Tactics (1)

- Ensure using tensor cores & correct format to avoid conversions/transposes
- Increase batch/grid sizes to more efficiently use the GPUs’ width
- Conventional parallelism
  - Increase framework’s worker threads (CLI args)
  - Python async, await, future, tasks, Dask, …
  - C/C++ OpenMP, OpenACC, pthreads, boost::asio, std::async, …
- Parallel pipelining - each stage can run in parallel
  - Are there a data dependencies between stages? No? Parallelize!
  - Ex: prefetch next batch/iteration of data while current batch/iteration is executing
  - Ex: within loader: net transfer, map, parse, load & transform, upload to GPU
- Reorder - could i do that sooner?
  - Prefetch - load data sooner & in parallel so it’s there before needed
- Fusing tiny kernels, copies, or memsets, or use cudaGraphs
- Overlap training (or inference)
  - CUDA MPS to share contexts & avoid switch overhead
  - Possibly difficult to fix both into memory.

Optimization Tactics (2)

- Pass buffers by pointer - avoid copies
- Multi-buffering - don’t make everyone wait on the same piece of memory
  - Often referred to as double or triple buffering; consider swap patterns
- Avoid returning data to CPU
- Avoid CPU pageable memory (prefer pinned / page-locked)
- Avoid unnecessary synchronizes
  - Avoid cuda*Synchronize functions, use cudaStreamWaitEvent instead
  - Avoid synchronous memory operation
  - Avoid CUDA default stream (if multi-stream)
- Pre-allocate memory, or use recycling tactics
- Minimize CUDA managed memory page faults on CPU & GPU (use prefetch)
- …



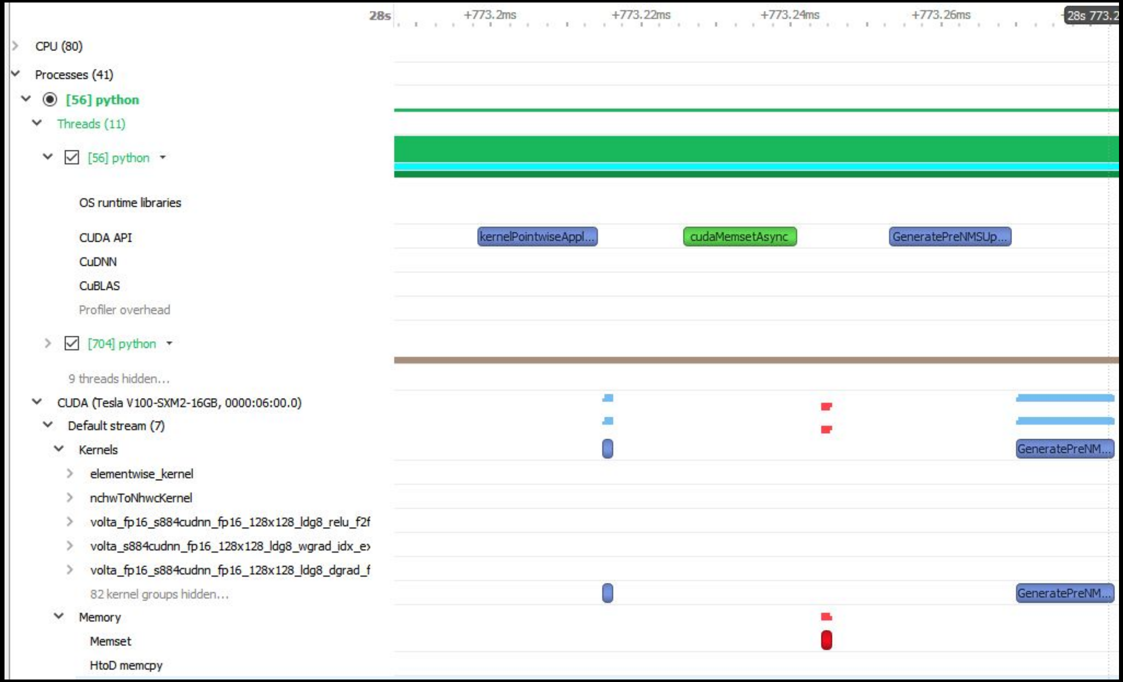
AWS Blog: Deepset achieves a 3.9x speedup and 12.8x cost reduction for training NLP models by working with AWS and NVIDIA

Deepset achieves a 3.9x speedup and 12.8x cost reduction for training NLP models by working with AWS and NVIDIA

https://aws.amazon.com/blogs/machine-learning/deepset-achieves-a-3-9x-speedup-and-12-8x-cost-reduction-for-training-nlp-models-by-working-with-aws-and-nvidia/

Cliff-notes

- Heavy use of Nsight Systems
- Switched from torch.nn.DataParallel (DP) to DistributedDataParallel (DDP)
- Enabled larger batch sizes by switching to Automatic Mixed Precision (AMP)
- Introduced a StreamingDataSilo & DALI to prefetch data
- …



Fusion opportunities

CPU launch cost + small GPU work size ≈ GPU sparse idle

This can apply to DNN nodes/layers

cudaMemcpyAsync behaving synchronous
Device to host pageable memory
Mitigate with pinned memory

Example GPU idle caused by stream synchronization

CPU & OS

For cases not caused by CUDA API usage or clarified by NVTX

Permissions

Some features are tied to OS permissions

- CPU thread state, core occupancy, user-space call-stack periodic sampling
  - Paranoid level too high
  - Container SECCOMP blocking perf_event_open
  - OS kernel samples require even lower paranoid levels and/or sudo
- ftrace
  - CAP_SYS_ADMIN | CAP_SYS_PERFMON

See online Nsight Systems documentation, and UI warnings
https://docs.nvidia.com/nsight-systems/InstallationGuide/index.html#linux-requirements
https://docs.nvidia.com/nsight-systems/UserGuide/index.html#docker-profiling

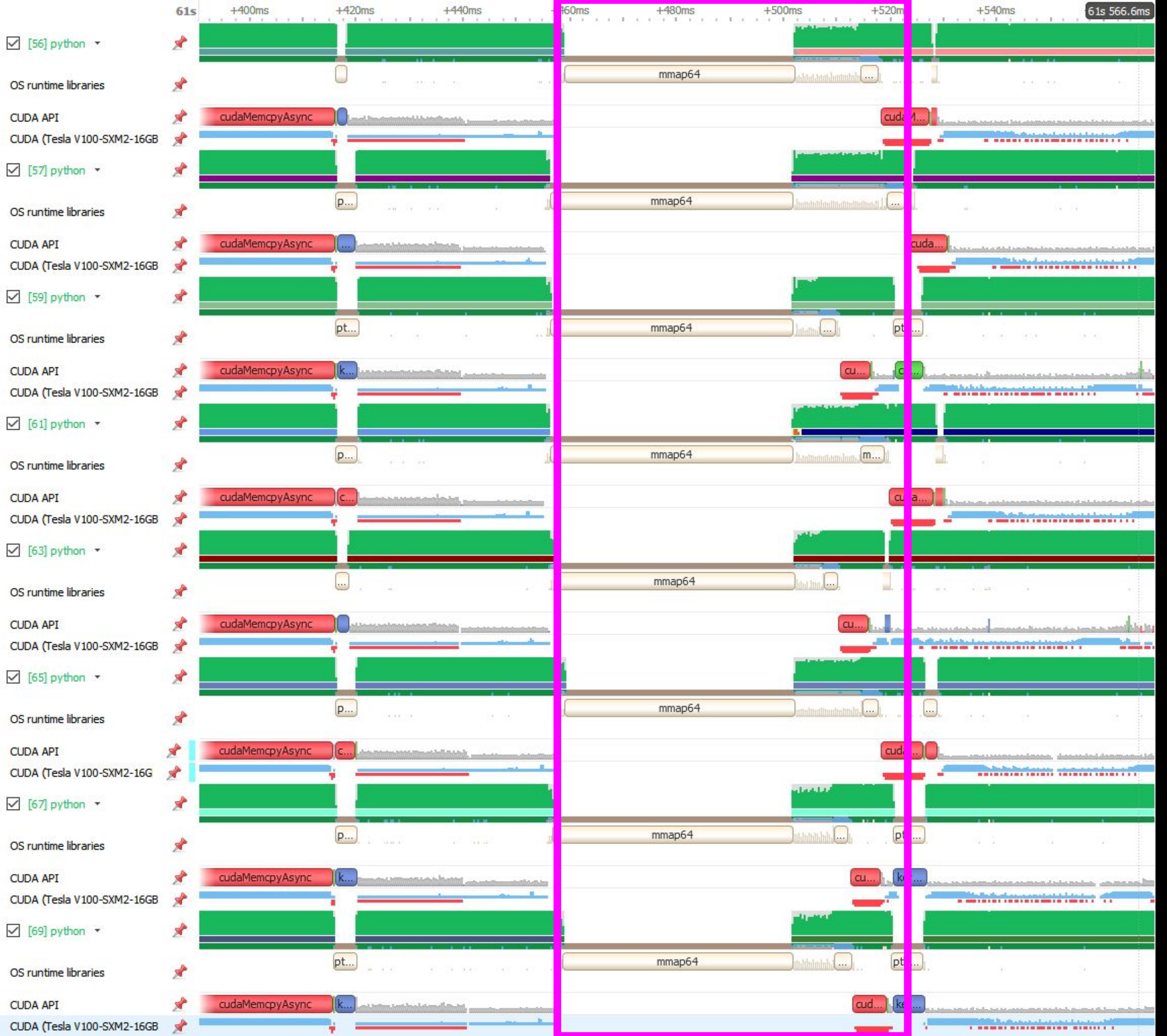
OS Runtime API Trace

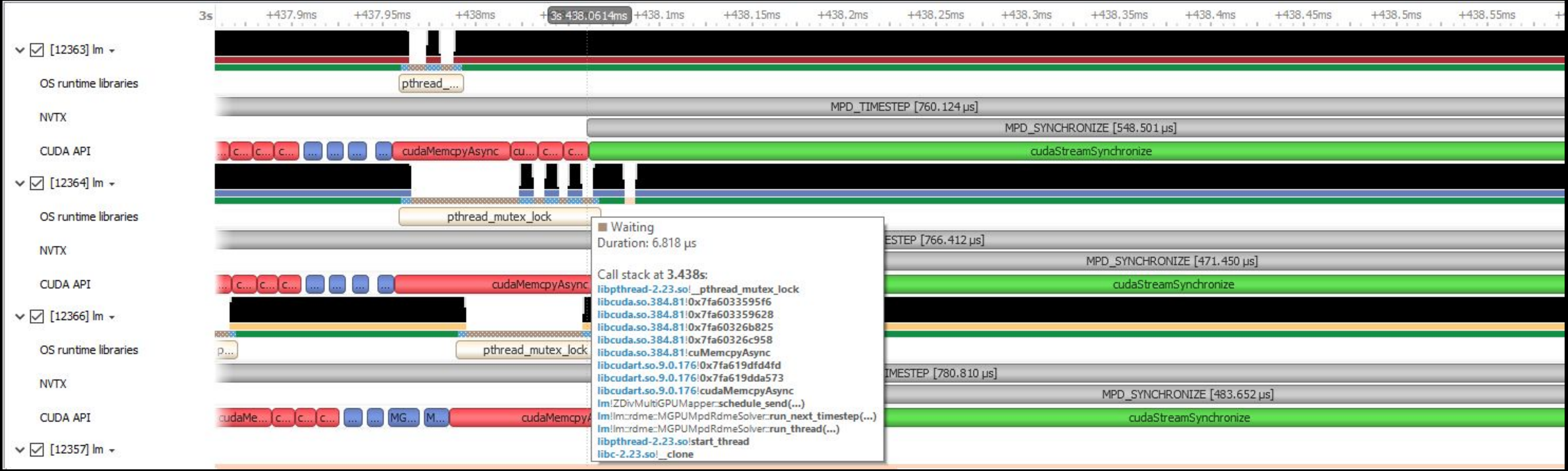
Example:Mask-RCNN

Map/unmap hiccups

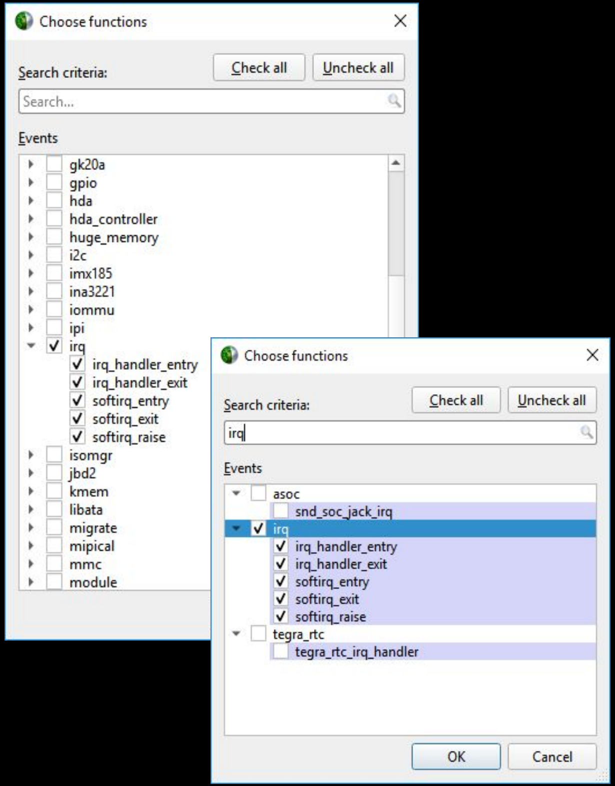
Mitigate by pipelining

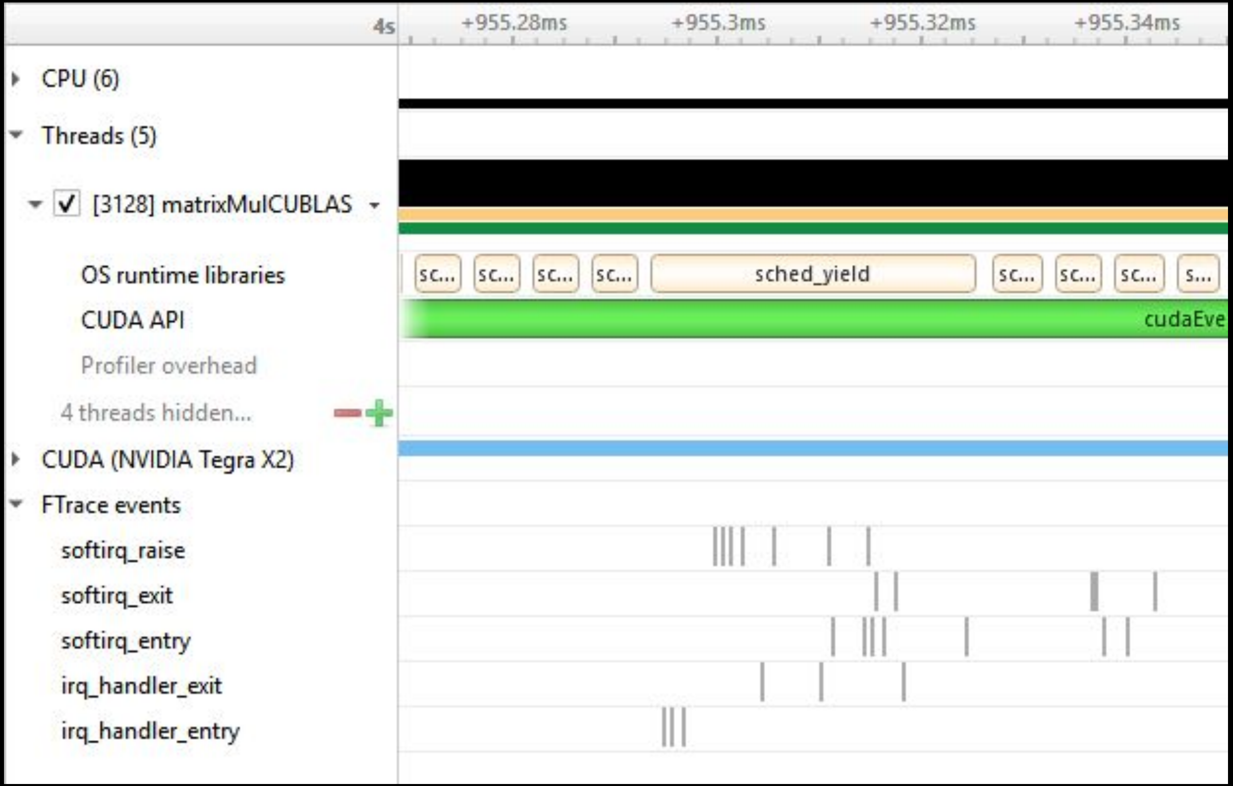
- Map 1 batch ahead
- Unmap last batch
- Swap pointers here instead





OS runtime trace (OSRT)
Includes backtraces of long running functions





FTrace

Example demonstrates interrupts

Bottom-Up View ▾ Process [9695] vmd_LINUXAMD64.11 (3 of 19 threads)

Filter... **99.82**% (23,260 samples) of data is shown due to applied filters.

Symbol Name	Self, %	Module Name
VolumetricData::compute_volume_gradient()	20.14	/home/johns/vmd/src/gtcbuilds/vmd_LINUXAMD64.11
VolumetricData::compute_volume_gradient()	20.14	/home/johns/vmd/src/gtcbuilds/vmd_LINUXAMD64.11
BaseMolecule::add_volume_data(char const*, double const*, double const*, double const*, double const*, int, int, int, float*)	18.30	/home/johns/vmd/src/gtcbuilds/vmd_LINUXAMD64.11
VMDApp::molecule_add_volumetric(int, char const*, double const*, double const*, double const*, double const*, int, int, int, float*)	18.30	/home/johns/vmd/src/gtcbuilds/vmd_LINUXAMD64.11
obj_segmentation(void*, Tcl_Interp*, int, Tcl_Obj* const*)	18.30	/home/johns/vmd/src/gtcbuilds/vmd_LINUXAMD64.11
[Max depth]	18.30	[Max depth]
BaseMolecule::add_volume_data(char const*, float const*, float const*, float const*, float const*, int, int, int, float*, float*, float*)	1.84	/home/johns/vmd/src/gtcbuilds/vmd_LINUXAMD64.11
MolFilePlugin::read_volumetric(Molecule*, int, int const*)	1.84	/home/johns/vmd/src/gtcbuilds/vmd_LINUXAMD64.11
VMDApp::molecule_load(int, char const*, char const*, FileSpec const*)	1.84	/home/johns/vmd/src/gtcbuilds/vmd_LINUXAMD64.11
text_cmd_mol(void*, Tcl_Interp*, int, char const**)	1.84	/home/johns/vmd/src/gtcbuilds/vmd_LINUXAMD64.11
TclInvokeStringCommand	1.84	/home/johns/vmd/src/gtcbuilds/vmd_LINUXAMD64.11
TclEvalObjvInternal	1.84	/home/johns/vmd/src/gtcbuilds/vmd_LINUXAMD64.11
TclExecuteByteCode	1.84	/home/johns/vmd/src/gtcbuilds/vmd_LINUXAMD64.11
TclCompEvalObj	1.84	/home/johns/vmd/src/gtcbuilds/vmd_LINUXAMD64.11
TclEvalObjEx	1.84	/home/johns/vmd/src/gtcbuilds/vmd_LINUXAMD64.11
Tcl_RecordAndEvalObj	1.84	/home/johns/vmd/src/gtcbuilds/vmd_LINUXAMD64.11
TclTextInterp::evalFile(char const*)	1.84	/home/johns/vmd/src/gtcbuilds/vmd_LINUXAMD64.11
VMDApp::logfile_read(char const*)	1.84	/home/johns/vmd/src/gtcbuilds/vmd_LINUXAMD64.11
VMDreadStartup(VMDApp*)	1.84	/home/johns/vmd/src/gtcbuilds/vmd_LINUXAMD64.11
[Max depth]	1.84	[Max depth]
0x7f10ca7022d6	5.13	/usr/lib64/libcuda.so.390.25
obj_segmentation(void*, Tcl_Interp*, int, Tcl_Obj* const*)	3.44	/home/johns/vmd/src/gtcbuilds/vmd_LINUXAMD64.11

Function table shows statistics from periodic call-stack backtraces

NVIDIA

CLI statistics and export

DL Rank Data Processing Imbalances

- Hurry up and wait
  - If anyone takes longer to reach the all-reduce then everyone is stuck!!!
- Load time
  - Did this batch take longer than the norm to load?
  - If parallel, did any rank have to wait?
- Processing time
  - Did this rank take longer than other
- Some remedies
  - Fix the data
    - 1 JPEG among PNGs or 1 MP3 among WAVs?
    - Wrong resolution, sample rate, precision, …
  - Reorganize your batch order
- A perfect job for scripts & statistics

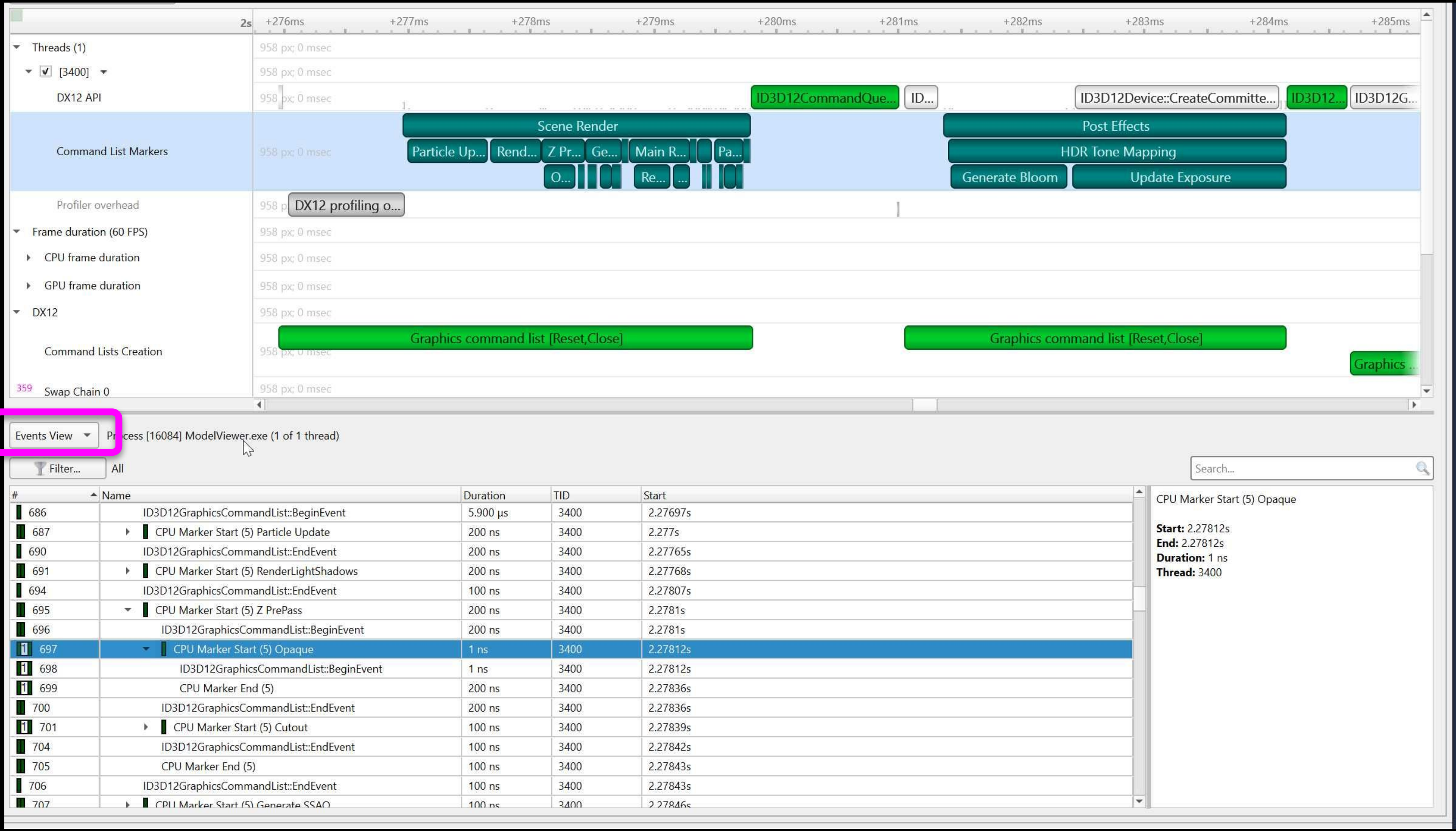
Time(%)	Time (ns)	Calls	Avg (ns)	Min (ns)	Max (ns)	Name
13.6	17198585879	66612	258190.5	20864	1327579	nhwc_batch_norm_fwd
11.3	14228434514	20075	708763.9	168352	1786335	nhwc_batch_norm_bwd_add_relu
10.4	13100639465	40158	326227.4	177824	1342749	volta_fp16_s884cudnn_fp16_128x128_ldg8_dgrad_f2f_exp_interior_nhwc_tt_v1
10.1	12688128728	42745	296833.1	168895	760641	volta_fp16_s884cudnn_fp16_128x128_ldg8_relu_f2f_exp_interior_nhwc_tn_v1
9.3	11744819167	41418	283568.0	167680	753631	volta_s884cudnn_fp16_128x128_ldg8_wgrad_idx_exp_interior_nhwc_nt
7.1	8935340081	23848	374678.8	229056	895710	volta_s884cudnn_fp16_64x64_sliced1x4_ldg8_wgrad_idx_exp_interior_nhwc_nt
6.2	7829371510	41408	189078.7	28800	1365372	nhwc_batch_norm_bwd_relu
4.1	5145597181	13820	372329.8	325313	524159	volta_fp16_s884cudnn_fp16_128x128_ldg8_relu_f2f_exp_small_nhwc_tn_v1
3.8	4796840892	12551	382187.9	311649	775264	volta_fp16_s884cudnn_fp16_128x128_ldg8_dgrad_f2f_exp_small_nhwc_tt_v1
2.9	3599110831	482226	7463.5	1760	868095	dctQuantInvJpegKernel
2.4	2968987604	5018	591667.5	124767	1418268	nhwc_batch_norm_bwd
2.0	2550050661	162488	15693.8	1184	1720191	ycbcr_to_format_kernel_roi
1.6	2031335889	3768	539101.9	381407	784829	dgrad_1x1_stride_2x2
1.5	1843345484	1256	1467631.8	1451800	1691492	pooling_bw_max_nhwc_kernel
1.2	1565496722	3781	414043.0	391712	480321	volta_fp16_s884cudnn_fp16_256x64_ldg8_relu_f2f_exp_small_nhwc_tn_v1
1.2	1543111896	3765	409857.1	386239	2808775	volta_fp16_s884cudnn_fp16_256x64_ldg8_dgrad_f2f_exp_small_nhwc_tt_v1
1.1	1373574502	2512	546805.1	434785	760834	dgrad_1d
1.0	1267659803	2514	504240.2	492895	601119	volta_fp16_s884cudnn_fp16_256x128_ldg8_relu_f2f_exp_small_nhwc_tn_v1
1.0	1215018985	58722	20691.0	1152	1325441	MapPlanKernel
0.8	1020510778	5670	179984.3	960	737985	mxnet_generic_kernel
0.7	909701850	65290	13933.2	6400	123808	convertTensor_kernel
0.7	894204599	2536	352604.3	192223	3158098	BatchedSeparableResampleKernel
0.7	868771063	1257	691146.4	670495	821372	first_layer_wgrad_kernel
0.7	859121367	1261	681301.6	675584	793889	first_layer_fwd_kernel
0.7	854624303	1257	679892.0	661086	726018	volta_fp16_s884cudnn_fp16_256x128_ldg8_relu_f2f_exp_interior_nhwc_tn_v1
0.6	719945062	1256	573204.7	555264	648066	dgrad_2d
0.4	479525740	6430	74576.3	2560	1835937	dcAcDecodeKernel
0.4	450010850	1268	354898.1	223551	1985854	BatchedCropMirrorNormalizePermuteKernel
0.3	436136496	1260	346140.1	342111	361025	pooling_fw_kernel_max_nhwc
0.3	364887793	1257	290284.6	164992	395295	volta_fp16_s884cudnn_fp16_256x64_ldg8_dgrad_f2f_exp_interior_nhwc_tt_v1
0.3	354605440	6430	55148.6	2176	1460961	transposeKernel
0.3	336891222	92	3661861.1	892835	26052414	convolve_common_engine_float_NHWC
0.2	302099259	65266	4628.7	928	48576	scalePackedTensor_kernel
0.2	277514711	123110	2254.2	1216	34304	computeOffsetsKernel
0.2	194972504	1261	154617.4	150848	177536	volta_fp16_s884cudnn_fp16_256x64_ldg8_relu_f2f_exp_interior_nhwc_tn_v1
0.1	176725421	46	3841857.0	578114	7942931	wgrad_alg0_engine_NHWC
0.1	160091745	65268	2452.8	1599	20384	computeWgradOffsetsKernel
0.1	116475012	1256	92734.9	91520	242912	pooling_bw_avg_nhwc_kernel
0.1	115380220	2159	53441.5	3008	1234110	destuffKernel
0.1	108445324	57731	1878.5	960	34016	computeBOffsetsKernel
0.1	77051216	23	3350052.9	1496867	7849586	wgrad_alg1_engine_NHWC
0.1	64827363	1256	51614.1	49663	67488	pooling_fw_4d_kernel

Stats/Export - CUDA kernel summary

Time(%)	Time (ns)	Instances	Avg (ns)	Min (ns)	Max (ns)	Range
4.0	162789746	70	2325567.8	248462	10421312	TensorRT:ExecutionContext::enqueue
0.4	16335803	24	680658.5	94707	2036983	TensorRT:predictions
0.3	10447691	70	149252.7	16094	3604518	TensorRT:conv1 + activation_1/Relu
0.2	10169055	70	145272.2	25936	1169143	TensorRT:conv1 + activation_1/Relu input reformatter 0
0.2	8698016	24	362417.3	9581	2970733	TensorRT:block_3b_conv_2
0.2	8625565	24	359398.5	28257	1242590	TensorRT:average_pooling2d_1
0.2	7733550	46	168120.7	30474	3498224	TensorRT:conv2d_cov/Sigmoid
0.2	6672310	70	95318.7	11271	1151050	TensorRT:block_1a_conv_1 + activation_2/Relu
0.1	5312256	70	75889.4	8316	2133255	TensorRT:block_4a_conv_2
0.1	5197452	24	216560.5	9572	1543570	TensorRT:block_3b_conv_1 + activation_12/Relu
0.1	4922521	70	70321.7	11732	1281643	TensorRT:block_1a_conv_shortcut + add_1 + activation_3/Relu
0.1	4151987	24	172999.5	10592	1625630	TensorRT:block_1b_conv_2
0.1	4121931	70	58884.7	10616	1307464	TensorRT:block_2a_conv_2
0.1	4108970	70	58699.6	9311	834374	TensorRT:block_1a_conv_2
0.1	4027181	24	167799.2	10564	1728250	TensorRT:block_1b_conv_shortcut + add_2 + activation_5/Relu
0.1	4018713	24	167446.4	9636	2431179	TensorRT:block_2b_conv_1 + activation_8/Relu
0.1	3799363	70	54276.6	8595	481048	TensorRT:block_3a_conv_2
0.1	3762886	24	156786.9	13969	1008096	TensorRT:block_4b_conv_shortcut + add_8 + activation_17/Relu
0.1	3376998	24	140708.2	12773	1641372	TensorRT:block_4b_conv_2
0.1	3361499	24	140062.5	11195	1248129	TensorRT:block_2b_conv_2
0.1	3227736	24	134489.0	15949	736287	TensorRT:predictions/Softmax
0.1	3023238	24	125968.2	11972	827474	TensorRT:block_2a_conv_shortcut + add_3 + activation_7/Relu
0.1	2998494	24	124937.2	12327	1217984	TensorRT:block_2a_conv_1 + activation_6/Relu
0.1	2853998	46	62043.4	10971	1983123	TensorRT:block_3a_conv_shortcut + add_3 + activation_7/Relu
0.1	2773491	24	115562.1	12019	809395	TensorRT:block_4a_conv_shortcut + add_7 + activation_15/Relu
0.1	2744711	24	114363.0	10474	892787	TensorRT:block_2b_conv_shortcut + add_4 + activation_9/Relu
0.1	2147200	24	89466.7	12037	645408	TensorRT:block_3a_conv_1 + activation_10/Relu
0.0	1914194	24	79758.1	10599	842947	TensorRT:block_4b_conv_1 + activation_16/Relu
0.0	1732011	24	72167.1	9659	488562	TensorRT:block_3a_conv_shortcut + add_5 + activation_11/Relu
0.0	1675232	24	69801.3	10073	623120	TensorRT:block_4a_conv_1 + activation_14/Relu
0.0	1525283	24	63553.5	10386	540733	TensorRT:block_3b_conv_shortcut + add_6 + activation_13/Relu
0.0	1358781	46	29538.7	10645	241099	TensorRT:block_3a_conv_1 + activation_6/Relu
0.0	1301126	46	28285.3	11929	244583	TensorRT:conv2d_cov

Stats/Export - NVTX code annotations

Note this includes TensorRT domains



Event Table & Statistics Table

Cluster

Multi-node

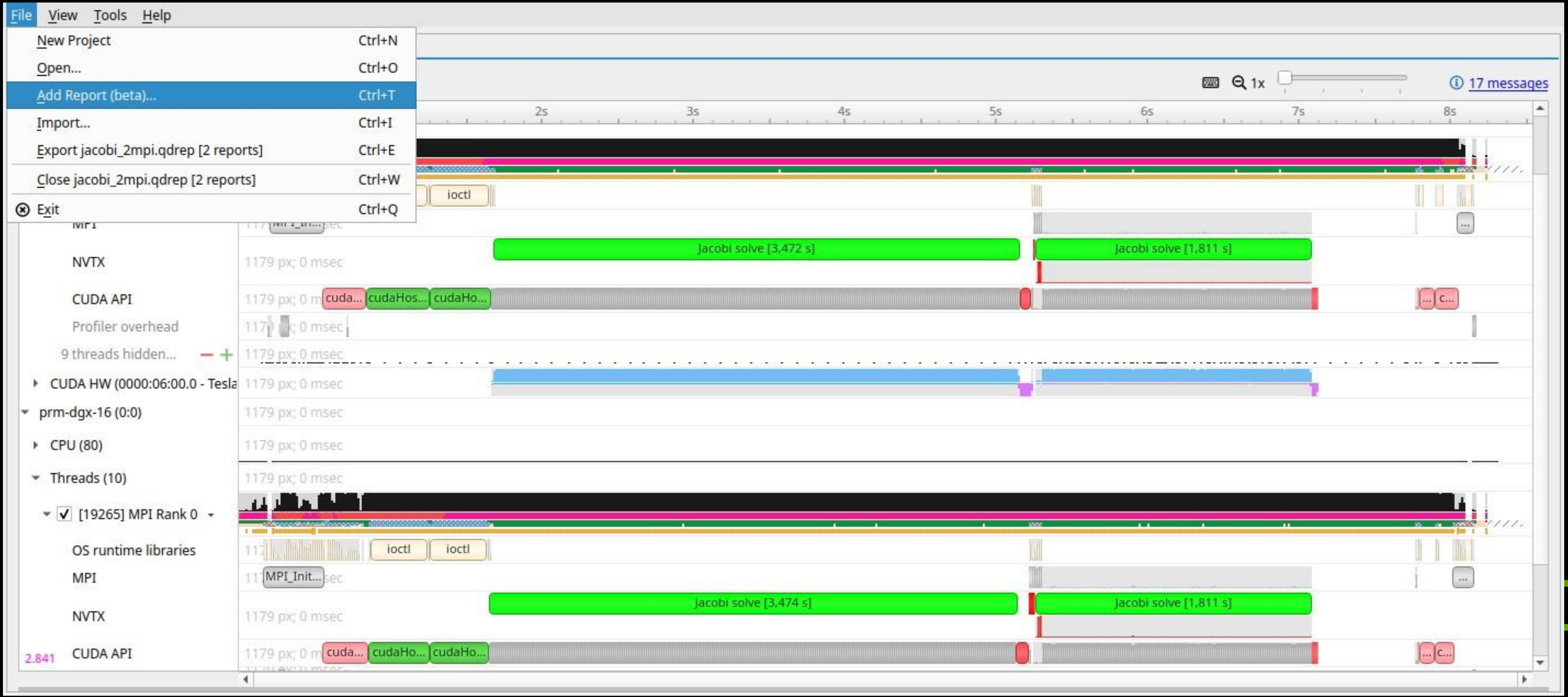
All the problems on a single node server (DONE)

Now you need to worry about

- Working with your cluster job/task scheduler
- Multi-report views
- But how do you pick which reports among thousands?
- Wall-clock time
- Networking
- Data analysis

Working with your cluster work scheduler

- srun <slurm_args> nsys profile <nsys_args> app <app_args>
- Make sure your report names are unique
  - --output=friendlyName_%q{SLURM_NODEID}_%q{SLURM_PROCID}.nsys-rep
- Avoid getting system-wide data from all ranks by moving nsys & app into a shell script
  - IF [$SLURM_LOCALID == 0] THEN
    - Add --nic-metrics=true
    - Add --gpu-metrics-device=all
- LIMIT your recording time otherwise you will run out of memory opening reports in the GUI on your laptop
  - 30sec-1min is often good enough. Or even less… capture a few iteration.
- I have more reports than I know what to do with. Now what?



Loading Multiple Reports into one Timeline

<= 2022.1 via file menu

>= 2022.2 via open dialog, select multiple report

NOTE: Pick a subset of reports otherwise you may run out of RAM

Wall-Clock Time

Timelines need it, especially if they are going to fit together accurately

Data is collected independently

Relies on the system’s wall clock time to be accurate

Otherwise there is timeline drift

From worst to best

- NTP = 1ms accuracy
- PTP software = 10 to 100 microseconds accuracy
- PTP hardware = can get down to ~10ns

Data Analysis

- Did not manually look at all files…. too many!
  - Maybe mix-n-match randomly with multi-report views?
- Use data analysis to hone in on interesting iterations → ranks → report
  - Cluster iteration times global total time
  - Per-rank iteration data load times
  - Per-rank iteration processing times
    - Total time to reach all-reduce
    - Forward pass
    - Backward pass
  - Per-rank iteration communication time, if not overlapped
  - DNN layer stats?
- Visually compare an average, min, & max report to try to understand how and why they differ

- You may need to add NVTX to your app to get some of this information
- GTC Talk: Scaling Transformer in PyTorch Across Multiple Nodes
  - https://developer.nvidia.com/gtc/2020/video/s21351
  - We'll make this type of stuff easier in the future

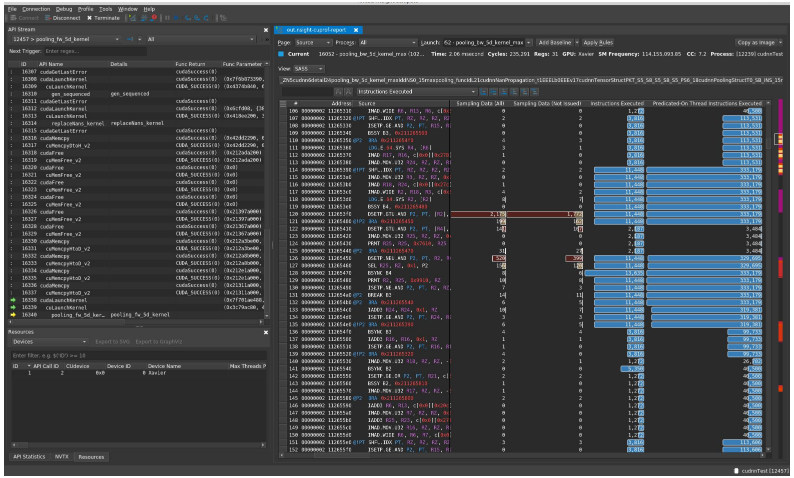
Data Processing Imbalances

- Make every rank consistent in the iteration
- Load time
    - Did this batch take longer than the norm to load?
    - If parallel, did any rank have to wait?
- Processing time
    - Did this batch take longer than the norm to process with the DNN?
- Hurry up and wait
    - If anyone takes longer to reach the all-reduce, everyone is stuck!!!
- Some remedies
    - Fix the data
        - A lonely JPEG in a PNG world, or MP3 among WAVs?
        - Wrong resolution, sample rate, precision, …
    - Reorganize your batch order

Other Products

Nsight Compute

CUDA Kernel Profiler



Interactive CUDA API debugger

Advanced CUDA Kernel Profiling

- CUDA-C/PTX/SASS correlation
- Source correlated performance metrics
- Diff'ing for performance reports
- Programmable expert system

NVTX-range-defined kernel profiling

High performance GUI visualization and CLI data collection

NOTE: See earlier slides about relationship with Nsight Systems! Start there to get big picture!

Windows 10, Linux Ubuntu 16.04/18.04/20.4, RHEL 7.x

Nsight Deep Learning Designer

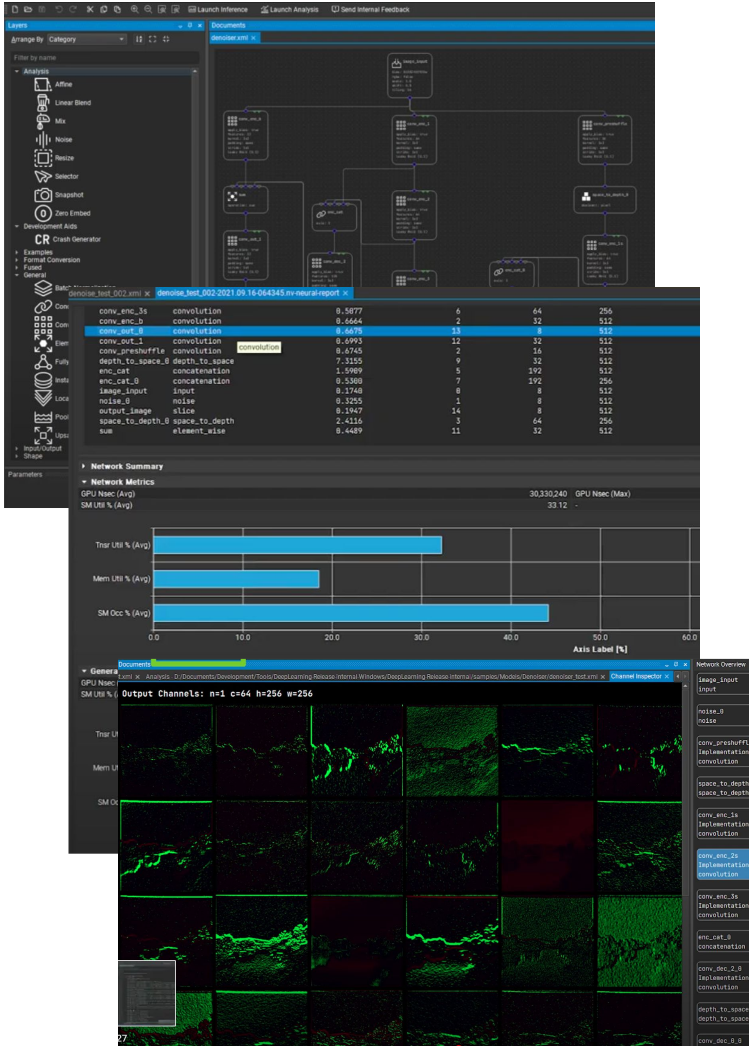
A new tools to add DL-based image processing feature to applications that have strict performance requirements

Designer, inspector, profiler image & video processing models

Download
https://developer.nvidia.com/nsight-dl-designer

Quick intro(2.5min) https://www.youtube.com/watch?v=7AraPM8dhyc

Extended Intro (35min)
https://www.nvidia.com/en-us/on-demand/session/siggraph2021-sigg21-s-05/?playlistId=playList-83726b46-d20a-40c2-a218-18c1b5fb0759



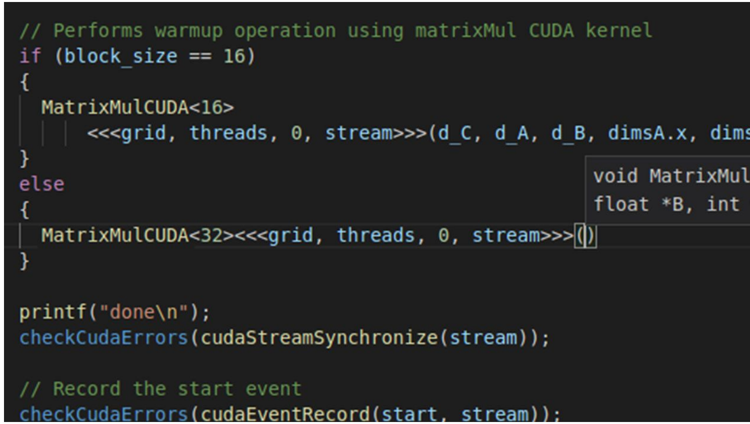
Nsight Visual Studio Code Edition

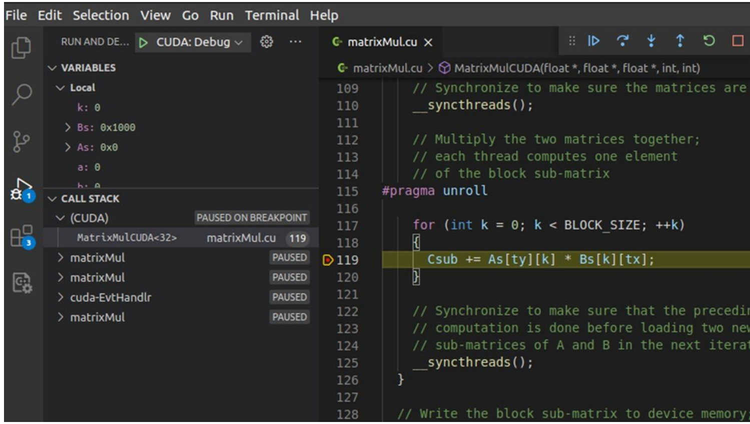
VSCode plugin for CUDA dev, compile, & debug

https://developer.nvidia.com/nsight-visual-studio-code-edition

Quick Intro (4min)
https://www.youtube.com/watch?v=gN3XeFwZ4ng

Microsoft Intro (14min)
https://www.youtube.com/watch?v=l6PgYhiQr-I





ADDITIONAL CUDA DEVELOPER TOOLS

Command-line CUDA tools

CUDA-gdb

Unified CPU and CUDA Debugging

CUDA-C/PTX/SASS support

Compute Sanitizer – API and utility

memcheck : reports out of bounds/misaligned memory access errors

racecheck : identifies races on `__shared__` memory

initcheck : usage of uninitialized global memory

synccheck : identify invalid usage of `__syncthreads()` and `__syncwarp()`

THANK YOU!

Download | **https://developer.nvidia.com/nsight-systems**

NOTE: Website version is newer than CUDA Toolkit

Let us know about your successes!