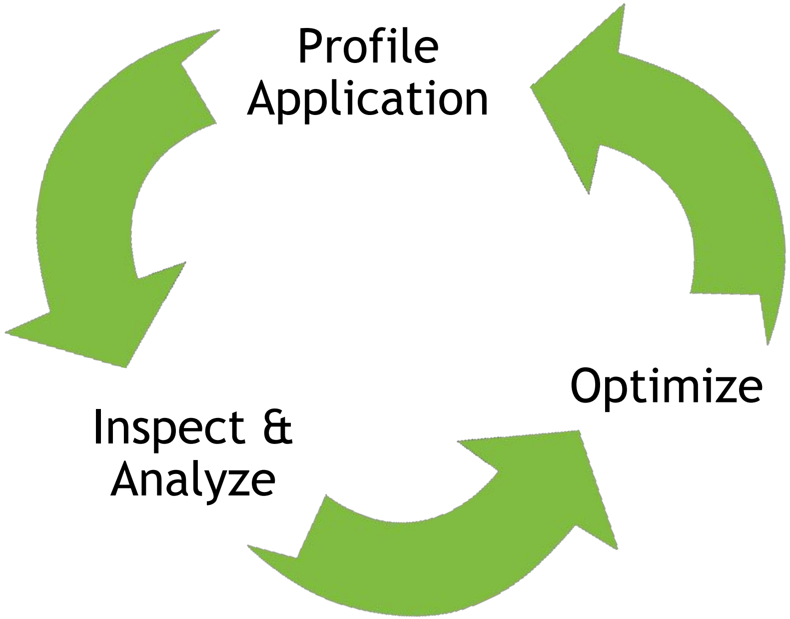
Nsight Systems – Introduction

Robert Dietrich – February 8, 2022

Typical Optimization Workflow



Iterate until desired performance is achieved

Legacy Transition

nvprof
Command-line tool

NVIDIA Visual Profiler
Standalone UI

Nsight Systems
Standalone GUI+CLI

CPU-GPU interactions & triage
Low overhead capture
GPU compute & graphics
Faster GUI + more data

Nsight Compute
Standalone GUI+CLI

GPU CUDA kernel analysis & debug
Very high freq GPU perf counters
Compare results (diff)
Incredible statistics & customizable

System-Wide Application Tuning

Maximize your GPU Investment

Locate optimization opportunities

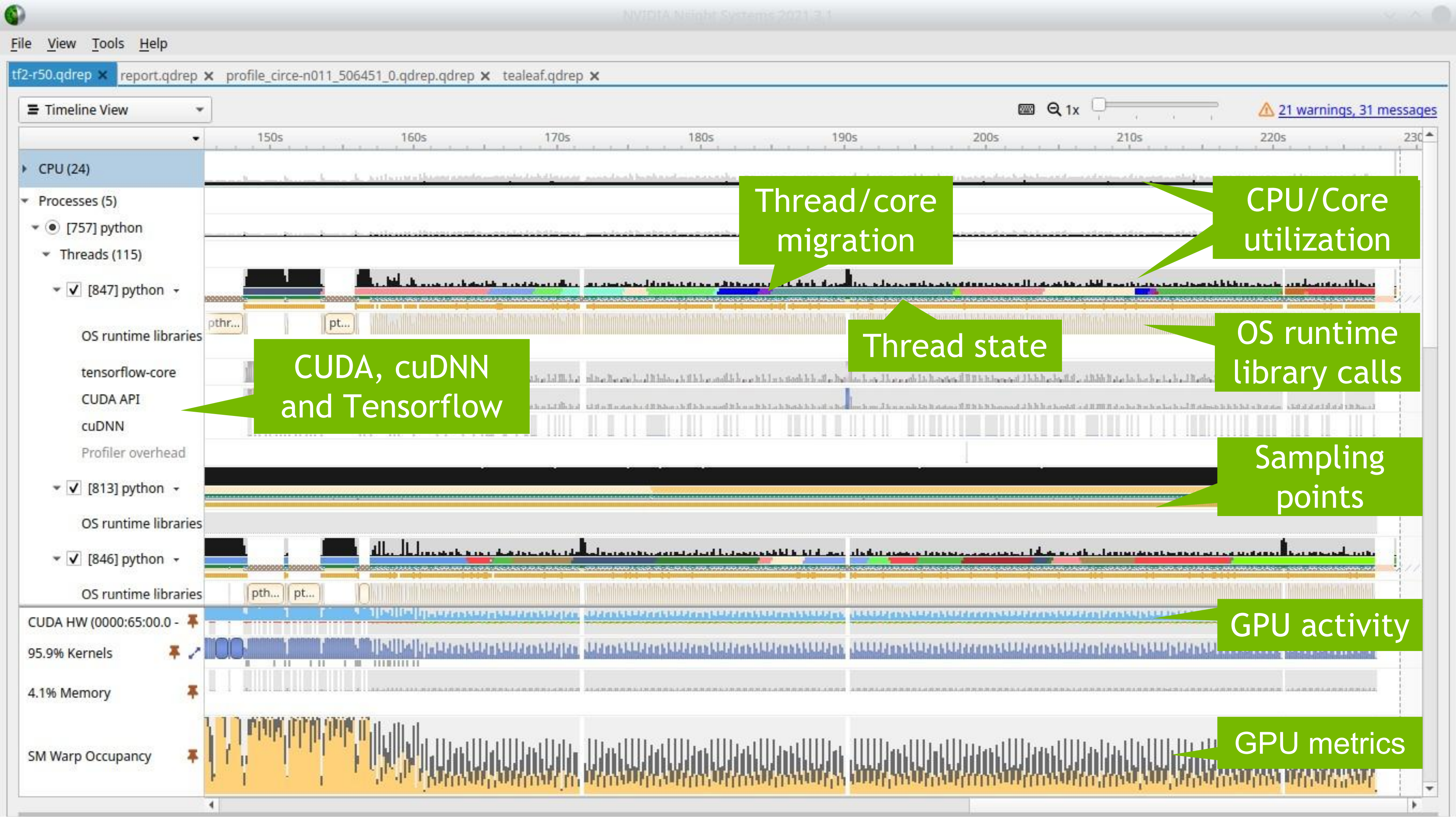
- Visualize millions of events on a timeline
- See gaps of unused CPU and GPU time

Balance your workload across multiple CPUs and GPUs

- CPU utilization and thread state
- GPU streams, kernels, memory transfers, etc.

Multi-platform support

- Linux, Windows and Mac OS X (host-only)
- x86-64, Power9, ARM server, Tegra (Linux & QNX)



File View Tools Help
tf2-r50.qdrep
report.qdrep
profile_circe-n011_506451_0.qdrep.qdrep
tealeaf.qdrep
Timeline View
1x
21 warnings, 31 messages
150s
160s
170s
180s
190s
200s
210s
220s
CPU (24)
Processes (5)
[757] python
Threads (115)
[847] python
OS runtime libraries
tensorflow-core
CUDA API
cuDNN
Profiler overhead
[813] python
OS runtime libraries
[846] python
OS runtime libraries
CUDA HW (0000:65:00.0 -
95.9% Kernels
4.1% Memory
SM Warp Occupancy
Thread/core migration
CPU/Core utilization
Thread state
OS runtime library calls
CUDA, cuDNN and Tensorflow
Sampling points
GPU activity
GPU metrics

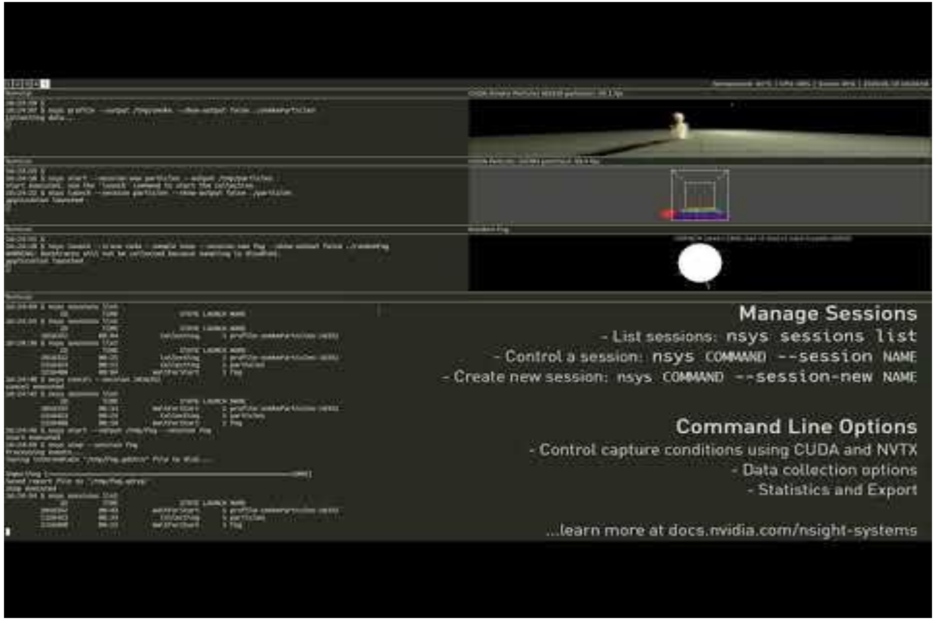
Command Line Interface (CLI)

Statistics and Export to SQLite, JSON, etc.

Command Line Interface

The Nsight Systems CLI provides several different commands

- Basic profiling session
  `nsys profile ./app`
- Interactive sessions (scriptable)
  `nsys start|launch|stop|cancel`
  `nsys session list`
  `nsys status|shutdown`
- Statistics and export
  `nsys stats|export`
  (export to sqlite, hdf, text, json, info)



https://youtu.be/r2ewwd4d0vc

CLI Profiling - Some Useful Switches

API tracing

```
-t, --trace=cuda,nvtx,osrt,opengl
(cublas,cusparse,cudnn,mpi,oshmem,ucx,openacc,openmp,vulkan,none)
```

Overwrite existing report

```
-f, --force-overwrite=[true|false]
```

Summary statistics (profile output on command line)

```
--stats=[true|false]
```

Report file name

```
-o, --output=report#
(patterns for hostname, PID and environment variables)
```

CLI Profiling - Some Useful Switches

Callstack sampling

`-s, --sample=[`**`cpu`**`|none]`

`--sampling-period=`*number of CPU Instructions Retired events*

`-b, --backtrace=[`**`lbr`**`|fp|dwarf|none]`

`--samples-per-backtrace={`**`1`**`..12}`

(The number of CPU IP samples collected for every CPU IP sample backtrace collected.)

Set the paranoid level: “`sudo sh -c 'echo 1 >/proc/sys/kernel/perf_event_paranoid’`

CUDA memory usage

`--cuda-memory-usage=[true|`**`false`**`]`

(Use `nsys profile --help` for a list of available options.)

CLI Profiling - MPI Programs

Single Node

`nsys profile [nsys_args] mpirun [mpirun_args] your_executable`

⇨ This creates one report file.

Multiple Nodes

`mpirun [mpirun_args] nsys profile [nsys_args] your_executable`

Set output report name with **`-o report_name_%q{OMPI_COMM_WORLD_RANK}`**
(for OpenMPI, **`PMI_RANK`** for MPICH and **`SLURM_PROCID`** for Slurm)

⇨ This creates one report file per MPI rank.

Profile only specific ranks. ⇨

```
#!/bin/bash
# OMPI_COMM_WORLD_LOCAL_RANK for node local rank
if [ $OMPI_COMM_WORLD_RANK -eq 0 ]; then
  nsys profile -t mpi "$@"
else
  "$@"
fi
```

CLI Profiling - Additional Output

```
WARNING: The command line includes a target application therefore the CPU context-switch
scope has been set to process-tree.
Collecting data…

  … APP OUTPUT …

Processing events...
Saving temporary "/tmp/nsys-report-2b96-9038-d0bd-2600.qdstrm" file to disk...

Creating final output files...
Processing [==============================================================100%]
Saved report file to "/tmp/nsys-report-2b96-9038-d0bd-2600.nsys-rep"
Report file moved to "/final/destination/report.nsys-rep"
```

Temporary data is written to **`/tmp/nvidia/nsight_systems`** by default. Set **`TMPDIR`** to specify another location.

CLI Stats Output

- CUDA API, kernels and memory operations (by time and by size)
- OS runtime
- NVTX

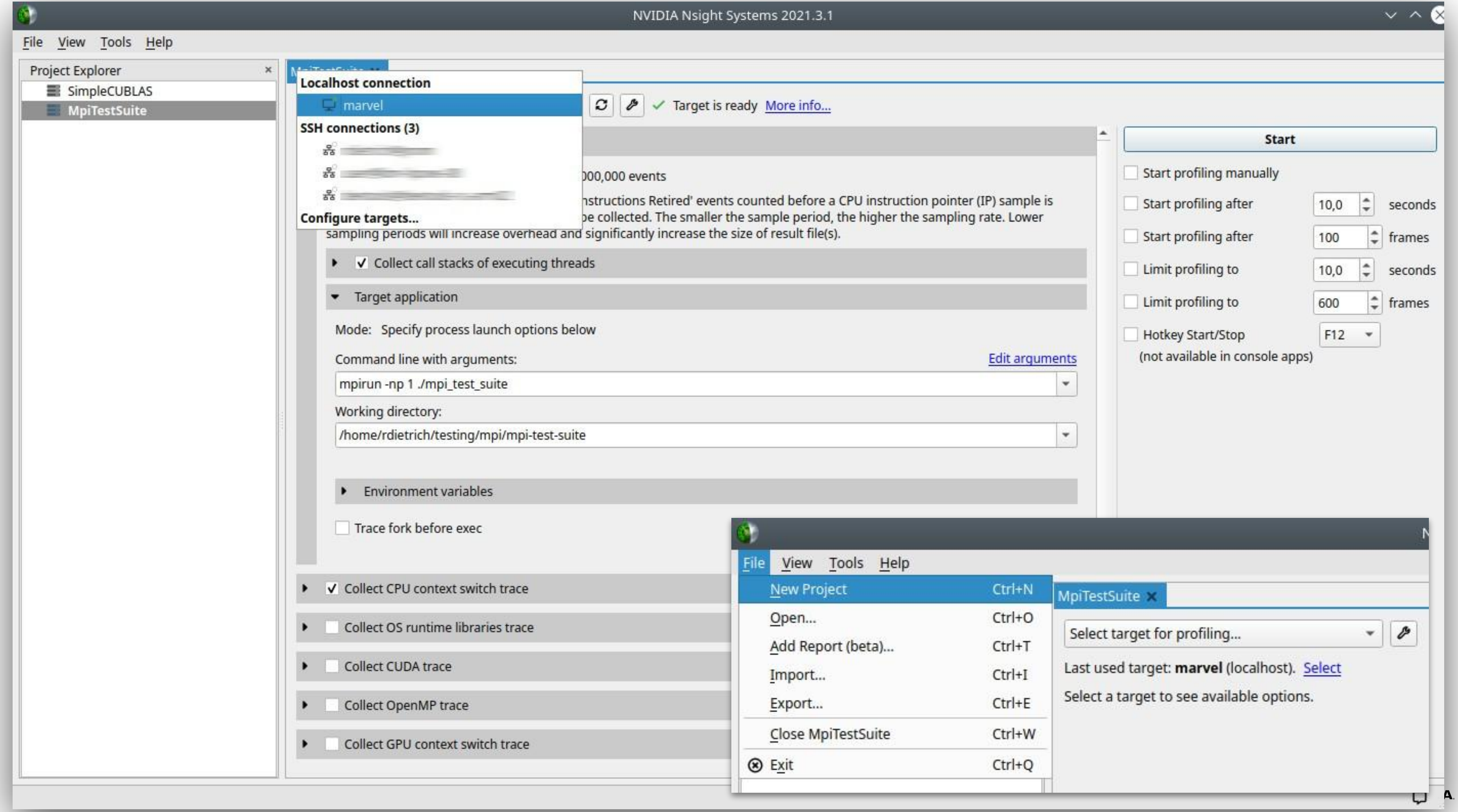
NVTX Push-Pop Range Statistics:

Time(%)	Total Time (ns)	Instances	Average	Minimum	Maximum	Range
80,0	16.193.376.534	21.268	761.396,0	6.823	2.438.848	MPI:MPI_Waitall
18,0	3.680.937.672	8.072	456.013,0	1.726	1.932.681	MPI:MPI_Allreduce
0,0	88.587.708	4	22.146.927,0	21.714.183	22.579.671	MPI:MPI_Init
0,0	57.888.066	21.268	2.721,0	1.319	297.466	MPI:MPI_Isend
0,0	24.282.234	21.268	1.141,0	872	16.931	MPI:MPI_Irecv
0,0	8.793.048	4	2.198.262,0	2.181.292	2.215.232	MPI:MPI_Finalize
0,0	3.814.158	28	136.219,0	2.302	1.265.786	MPI:MPI_Barrier
0,0	266.630	32	8.332,0	832	104.980	MPI:MPI_Reduce

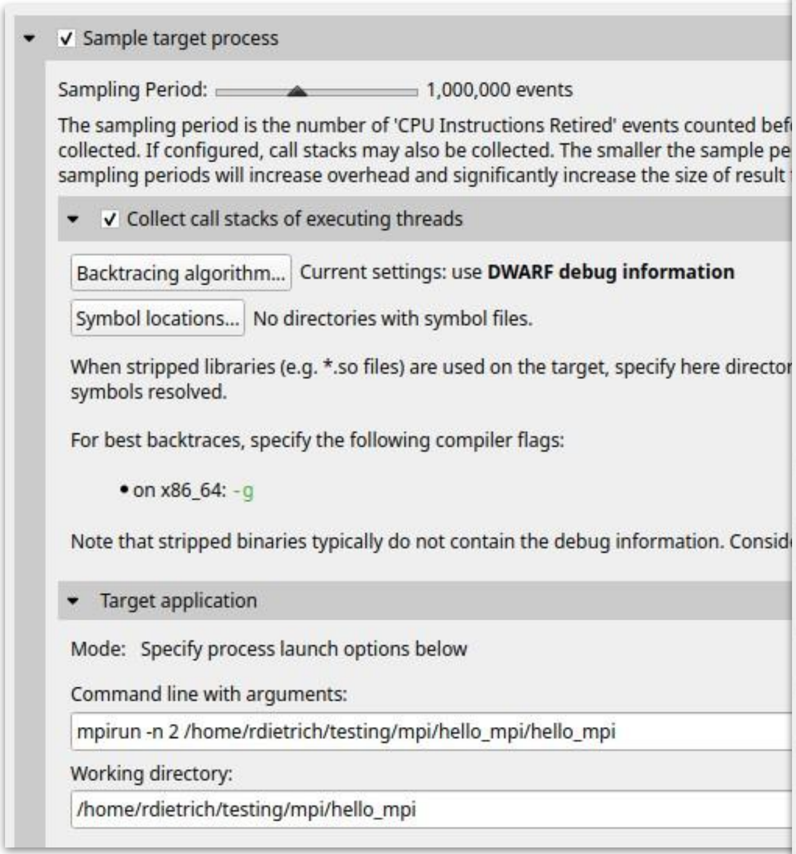
CUDA Kernel Statistics:

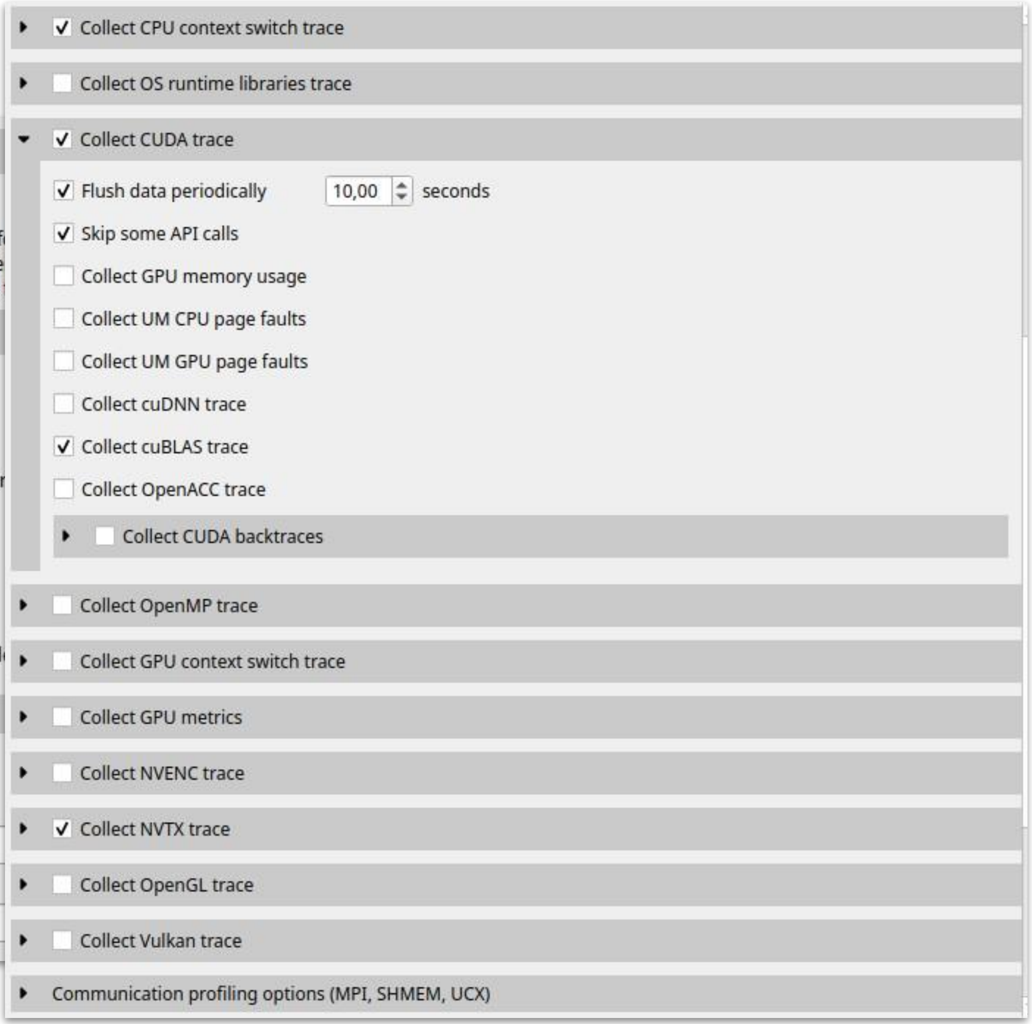
Time(%)	Total Time (ns)	Instances	Average	Minimum	Maximum	Name
37,0	7.344.350.347	8.568	857.183,0	851.546	876.922	device_tea_leaf_ppcg_solve_calc_sd_new(kernel_info_t, dou
36,0	7.229.217.310	8.568	843.746,0	839.323	899.482	device_tea_leaf_ppcg_solve_update_r(kernel_info_t, double
10,0	2.050.961.573	2.010	1.020.378,0	1.005.849	1.036.569	device_tea_leaf_cg_solve_calc_ur(kernel_info_t, double, d
9,0	1.879.919.365	2.010	935.283,0	913.754	1.214.808	device_tea_leaf_cg_solve_calc_w(kernel_info_t, double*, d
5,0	1.008.596.179	1.980	509.392,0	501.469	521.756	device_tea_leaf_cg_solve_calc_p(kernel_info_t, double, do
0,0	61.606.046	30	2.053.534,0	1.176.600	2.890.541	device_tea_leaf_calc_rrn(kernel_info_t, double const*, do
0,0	38.094.843	12.336	3.088,0	1.887	13.184	void reduction<double, (REDUCTION_TYPE)0>(int, double*)

Launch Application via GUI

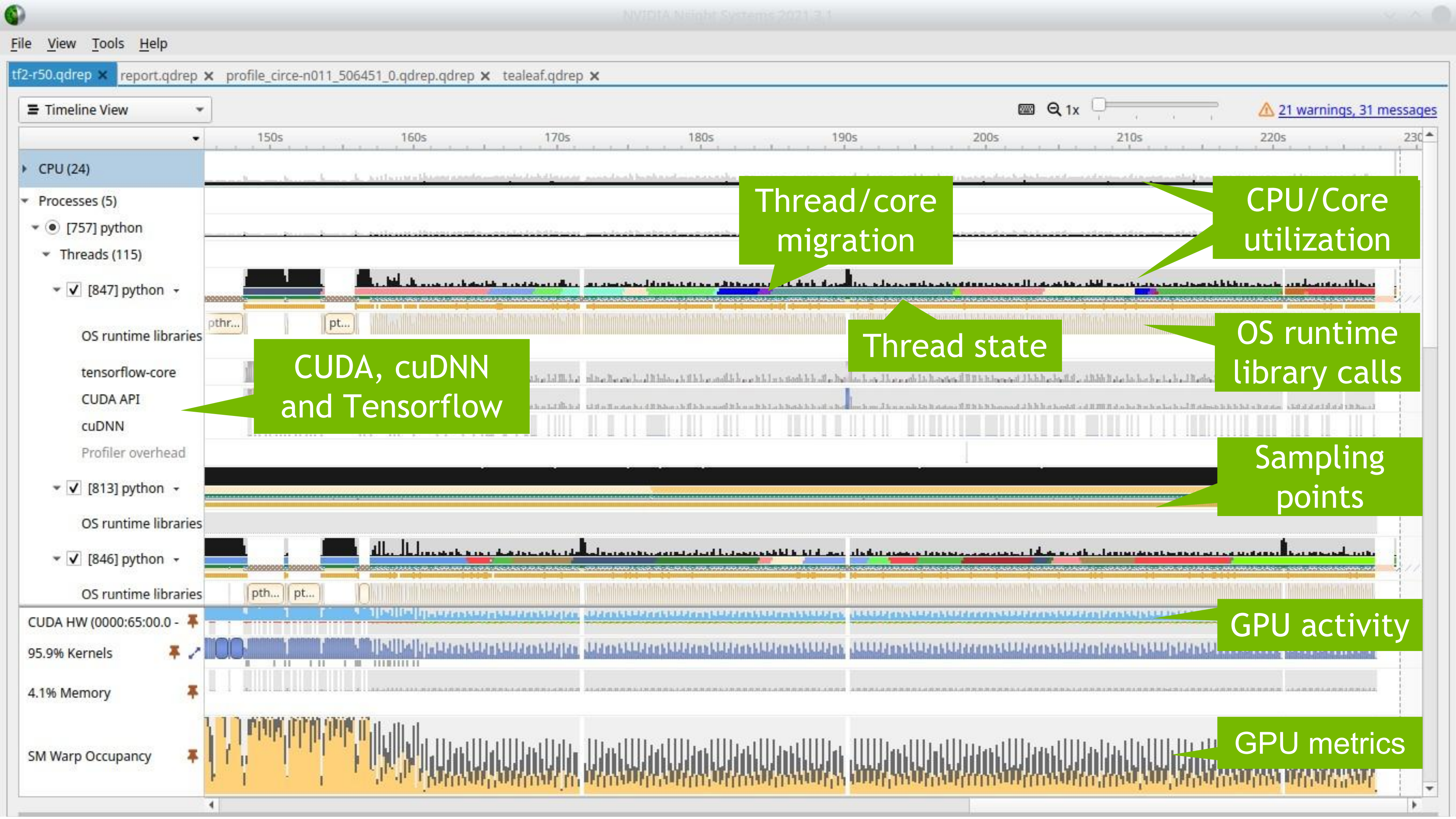

NVIDIA Nsight Systems 2021.3.1
File View Tools Help
Project Explorer
SimpleCUBLAS
MpiTestSuite
Localhost connection
marvel
SSH connections (3)
Configure targets...
Target is ready More info...
Start
Start profiling manually
Start profiling after 10,0 seconds
Start profiling after 100 frames
Limit profiling to 10,0 seconds
Limit profiling to 600 frames
Hotkey Start/Stop F12
(not available in console apps)
000,000 events
nstructions Retired' events counted before a CPU instruction pointer (IP) sample is
be collected. The smaller the sample period, the higher the sampling rate. Lower
sampling periods will increase overhead and significantly increase the size of result file(s).
Collect call stacks of executing threads
Target application
Mode: Specify process launch options below
Command line with arguments:
Edit arguments
mpirun -np 1 ./mpi_test_suite
Working directory:
/home/rdietrich/testing/mpi/mpi-test-suite
Environment variables
Trace fork before exec
Collect CPU context switch trace
Collect OS runtime libraries trace
Collect CUDA trace
Collect OpenMP trace
Collect GPU context switch trace
File View Tools Help
New Project Ctrl+N
Open... Ctrl+O
Add Report (beta)... Ctrl+T
Import... Ctrl+I
Export... Ctrl+E
Close MpiTestSuite Ctrl+W
Exit Ctrl+Q
MpiTestSuite
Select target for profiling...
Last used target: marvel (localhost). Select
Select a target to see available options.

Feature Selection





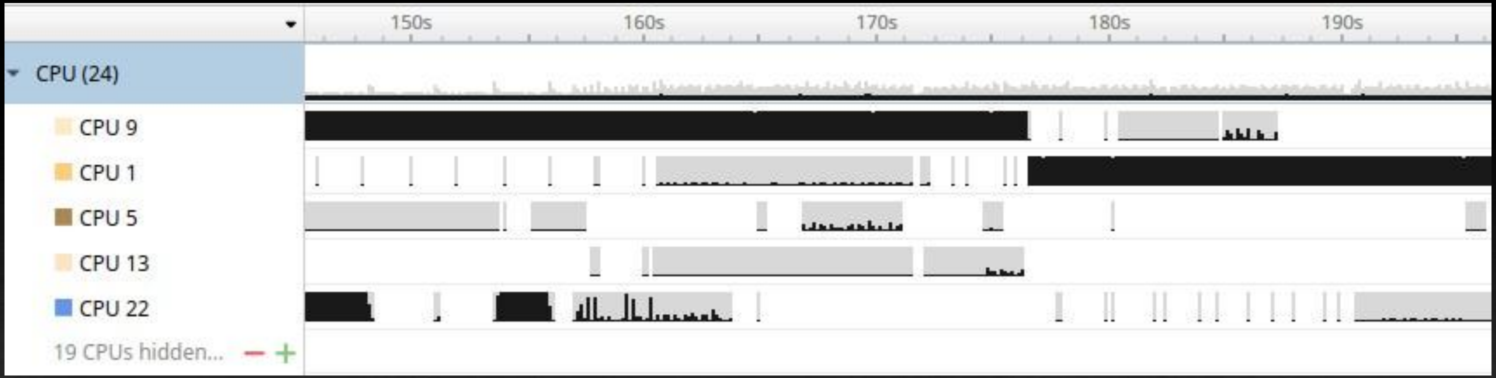
Feature Highlights


File View Tools Help
tf2-r50.qdrep
report.qdrep
profile_circe-n011_506451_0.qdrep.qdrep
tealeaf.qdrep
Timeline View
1x
21 warnings, 31 messages
150s
160s
170s
180s
190s
200s
210s
220s
CPU (24)
Processes (5)
[757] python
Threads (115)
[847] python
OS runtime libraries
tensorflow-core
CUDA API
cuDNN
Profiler overhead
[813] python
OS runtime libraries
[846] python
OS runtime libraries
CUDA HW (0000:65:00.0 -
95.9% Kernels
4.1% Memory
SM Warp Occupancy
Thread/core migration
CPU/Core utilization
Thread state
OS runtime library calls
CUDA, cuDNN and Tensorflow
Sampling points
GPU activity
GPU metrics

CPU Cores Workload

See CPU core utilization by application's threads

Locate idle time on CPU cores

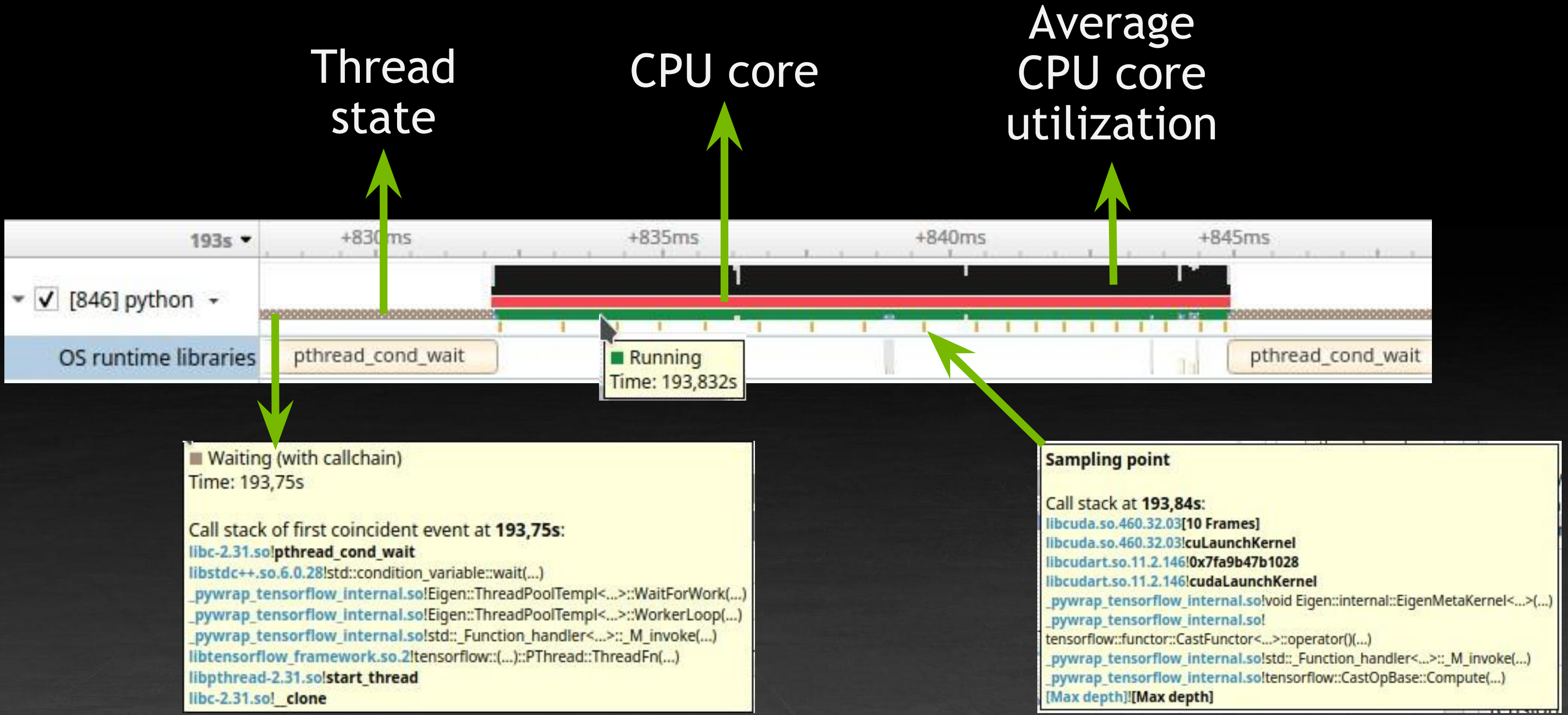


Time ruler

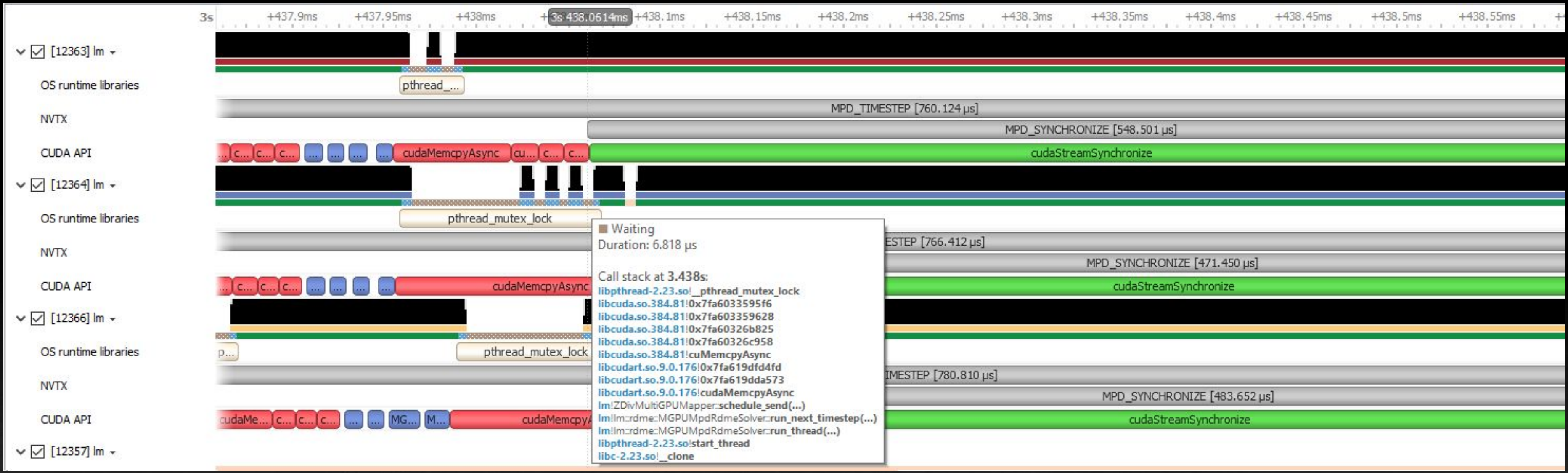
Average utilization

Cores Color

CPU Thread Activity



OS Runtime Libraries

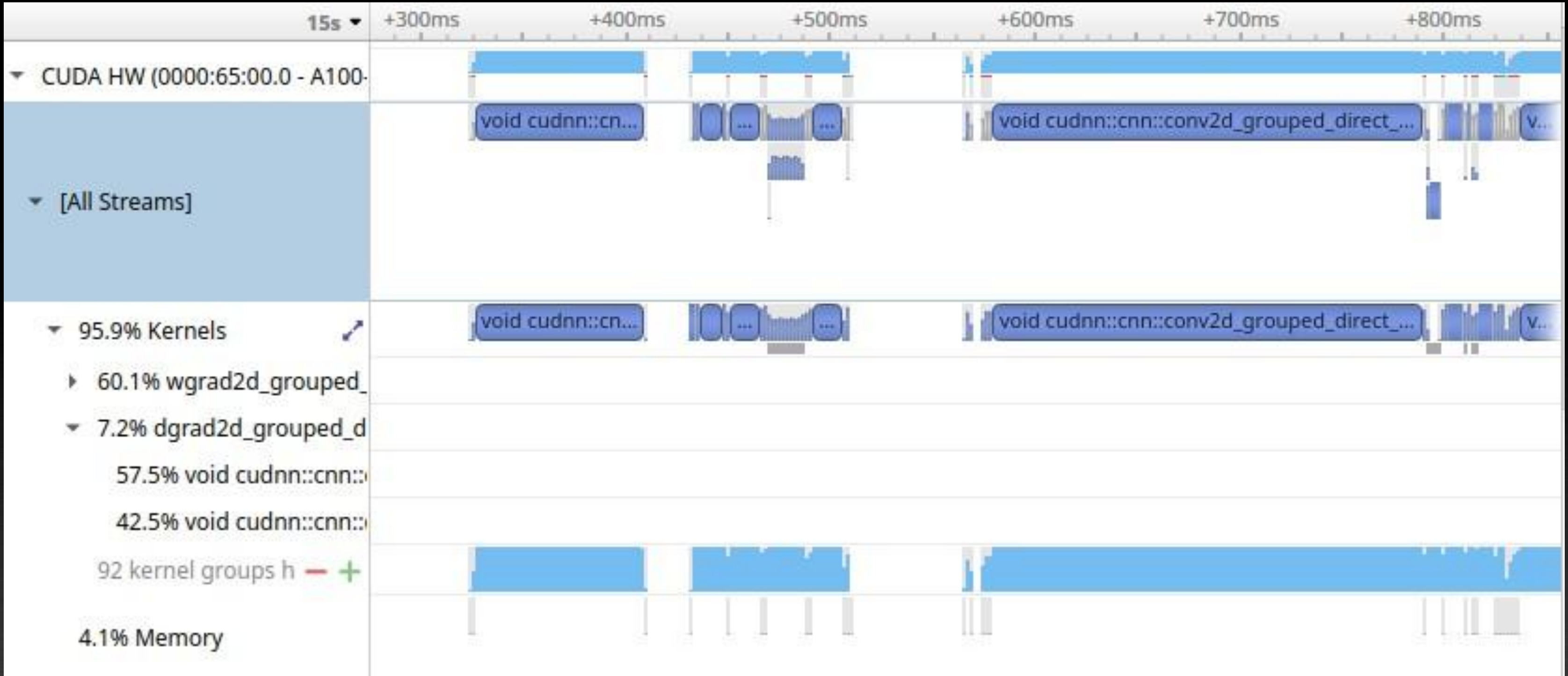


Includes backtraces of long running functions

CUDA API

- Trace CUDA API Calls on OS thread
- See when kernels are dispatched
- See when memory operations are initiated
- Locate the corresponding CUDA workload on GPU

GPU Utilization



GPU Utilization

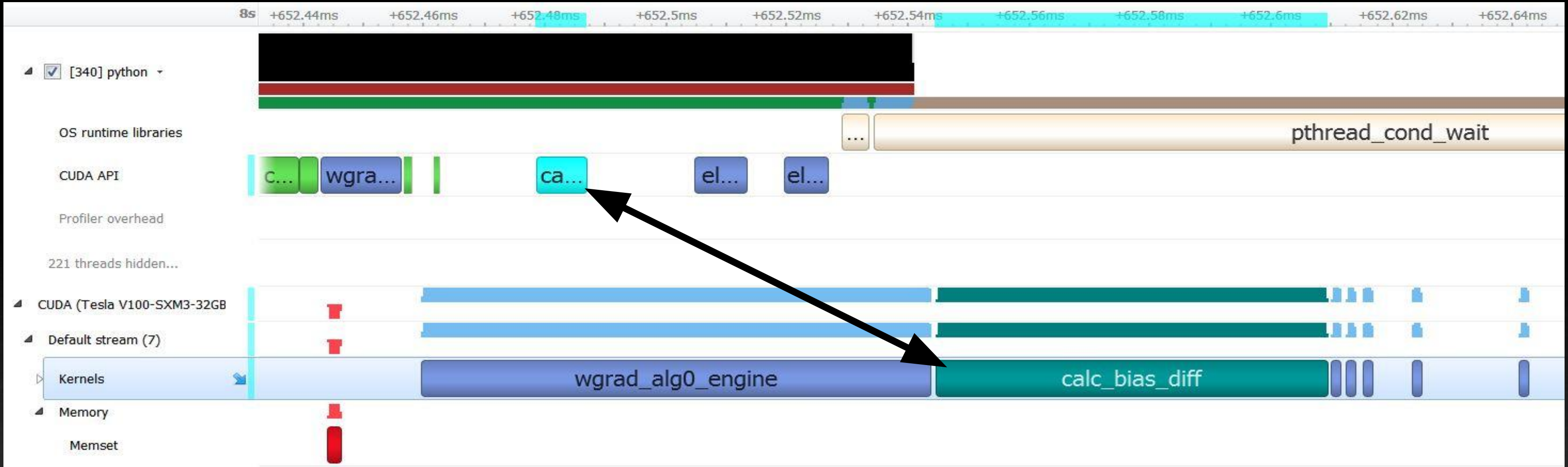
avg. CUDA kernel coverage

avg. no. of memory operations

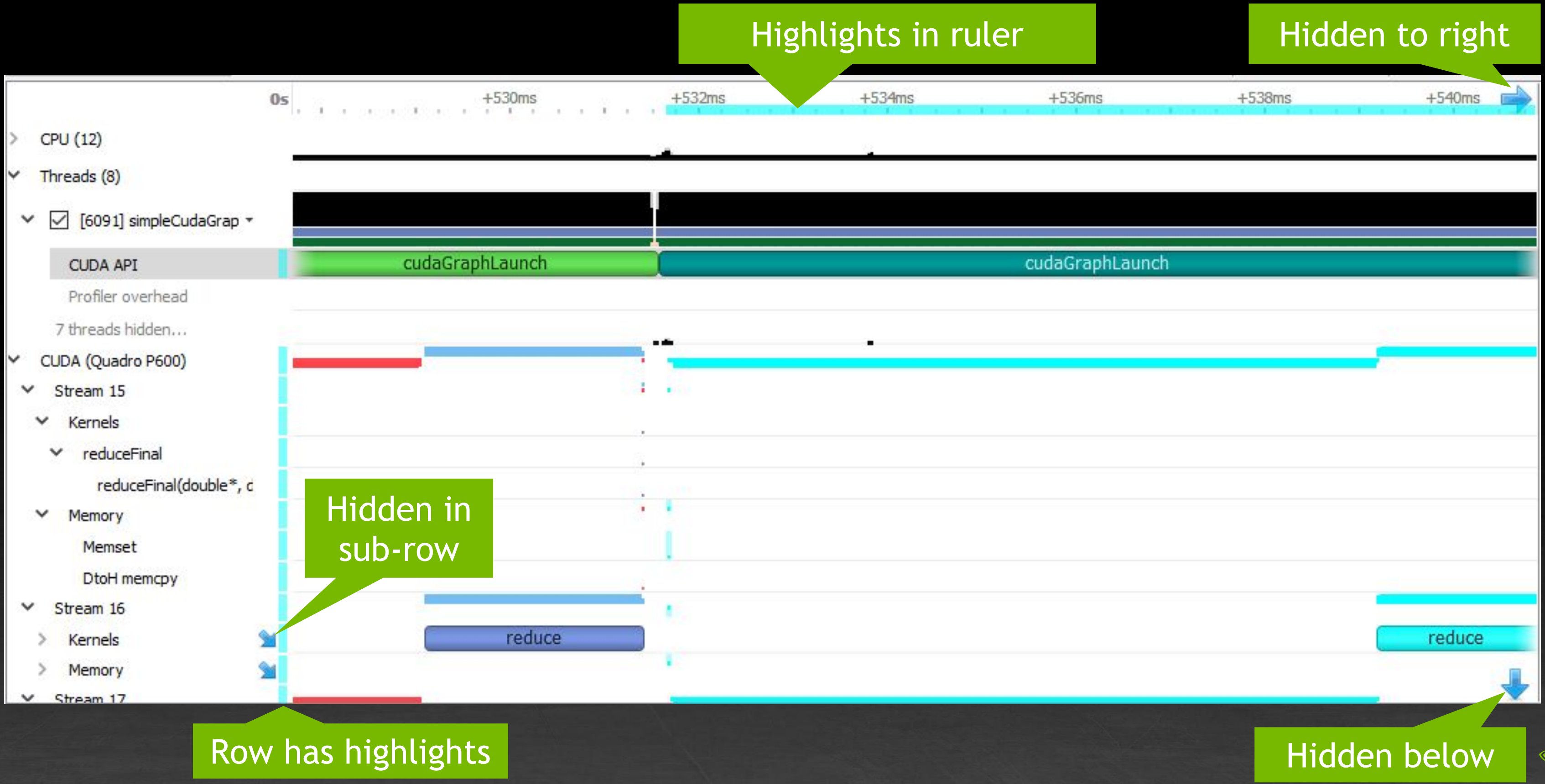
Zooming in

… reveals gaps where there were valleys.

CPU-GPU Correlation

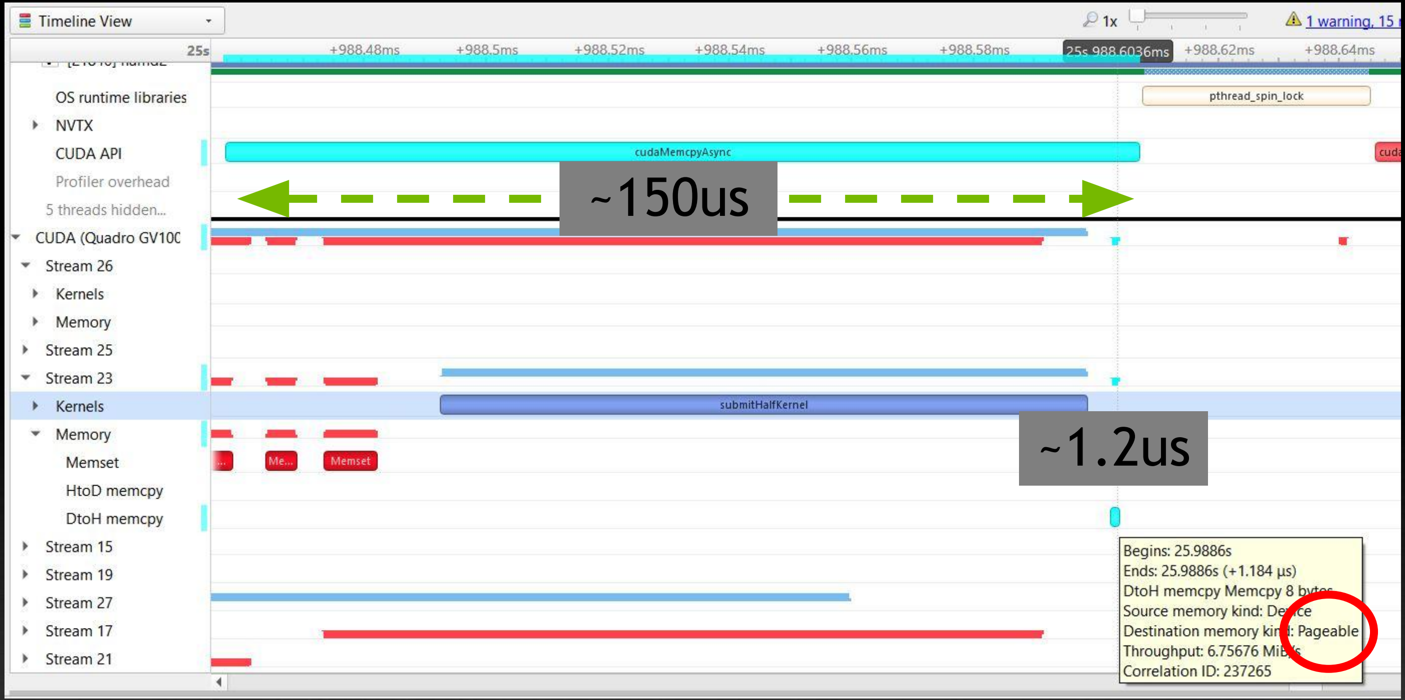


CPU-GPU Correlation & Location Assistance

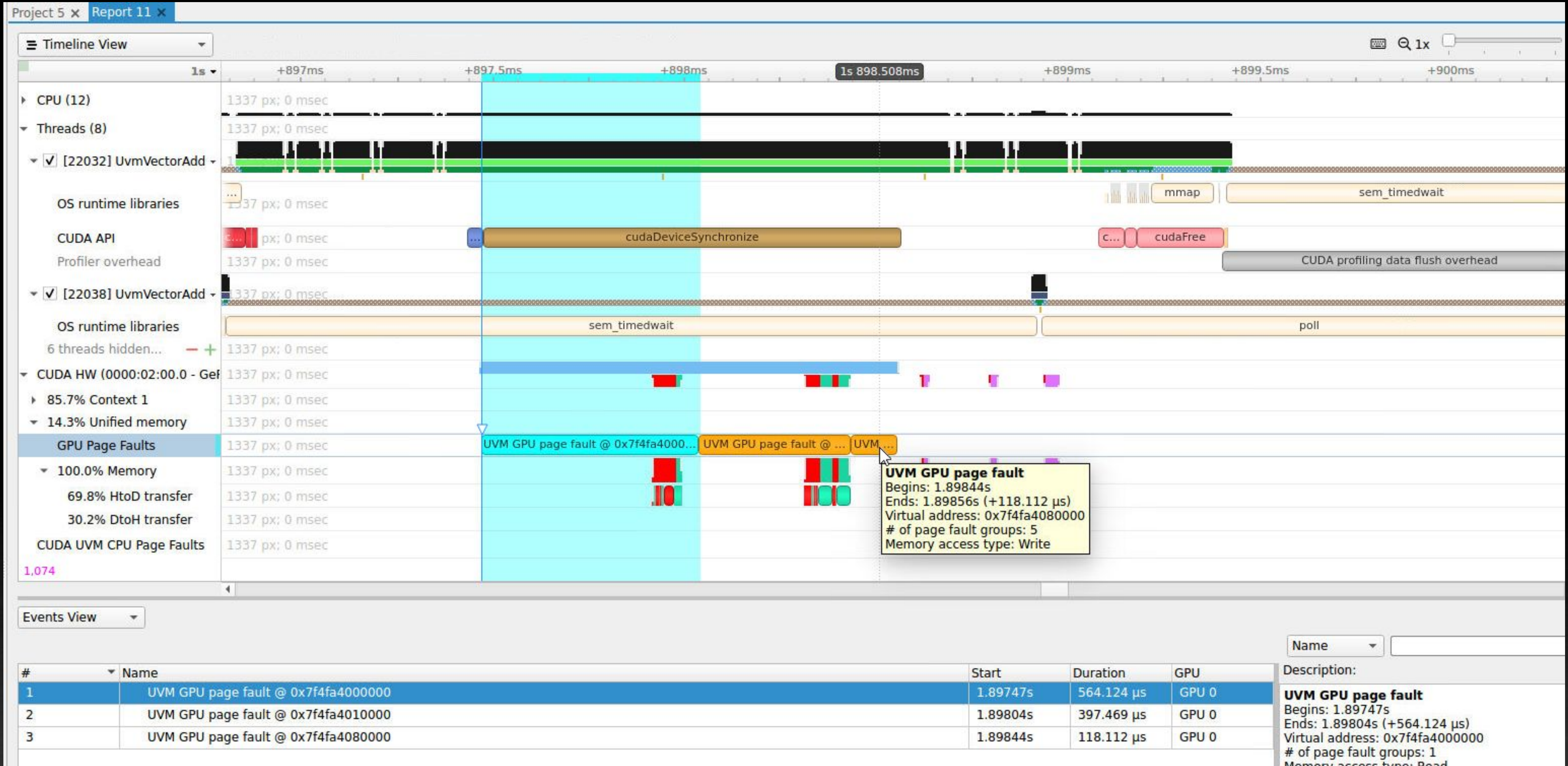


NVIDIA

CUDA Memory Transfers



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



CUDA Unified Memory CPU & GPU Page Fault Trace

CUDA Libraries & DL Frameworks

cuDNN, cuBLAS, cuSPARSE & TensorRT, Tensorflow

Host operation is mapped to GPU operations

User annotations APIs for CPU & GPU

NVTX, OpenGL, Vulkan, and Direct3D performance markers

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

Additional Views

Statistical Sampling

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.

IP sample

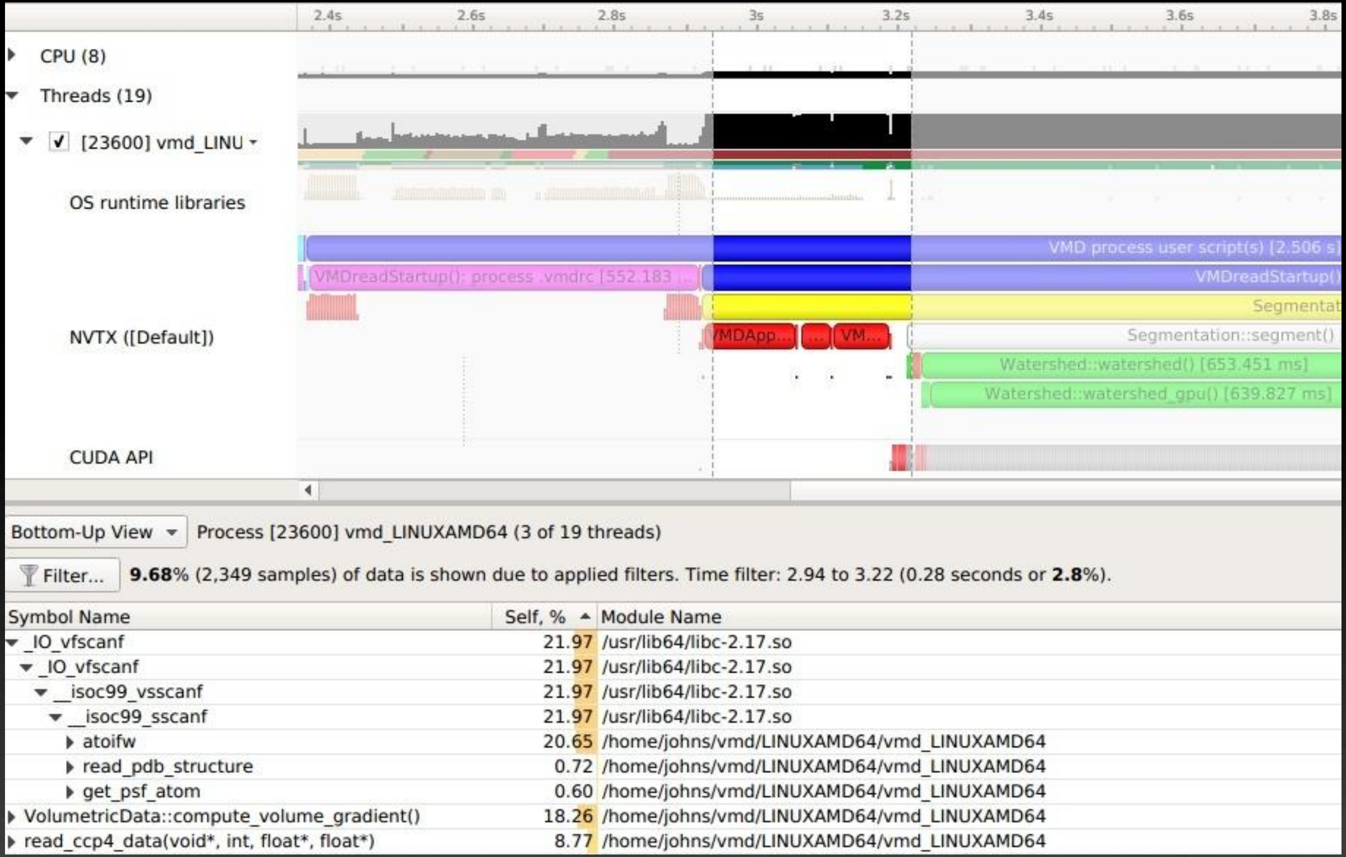
Code path 1

Code path 2 leading to IP sample

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

shows statistics from periodic call-stack backtraces

Statistical Sampling - Filtering



Events View

Events View

Name

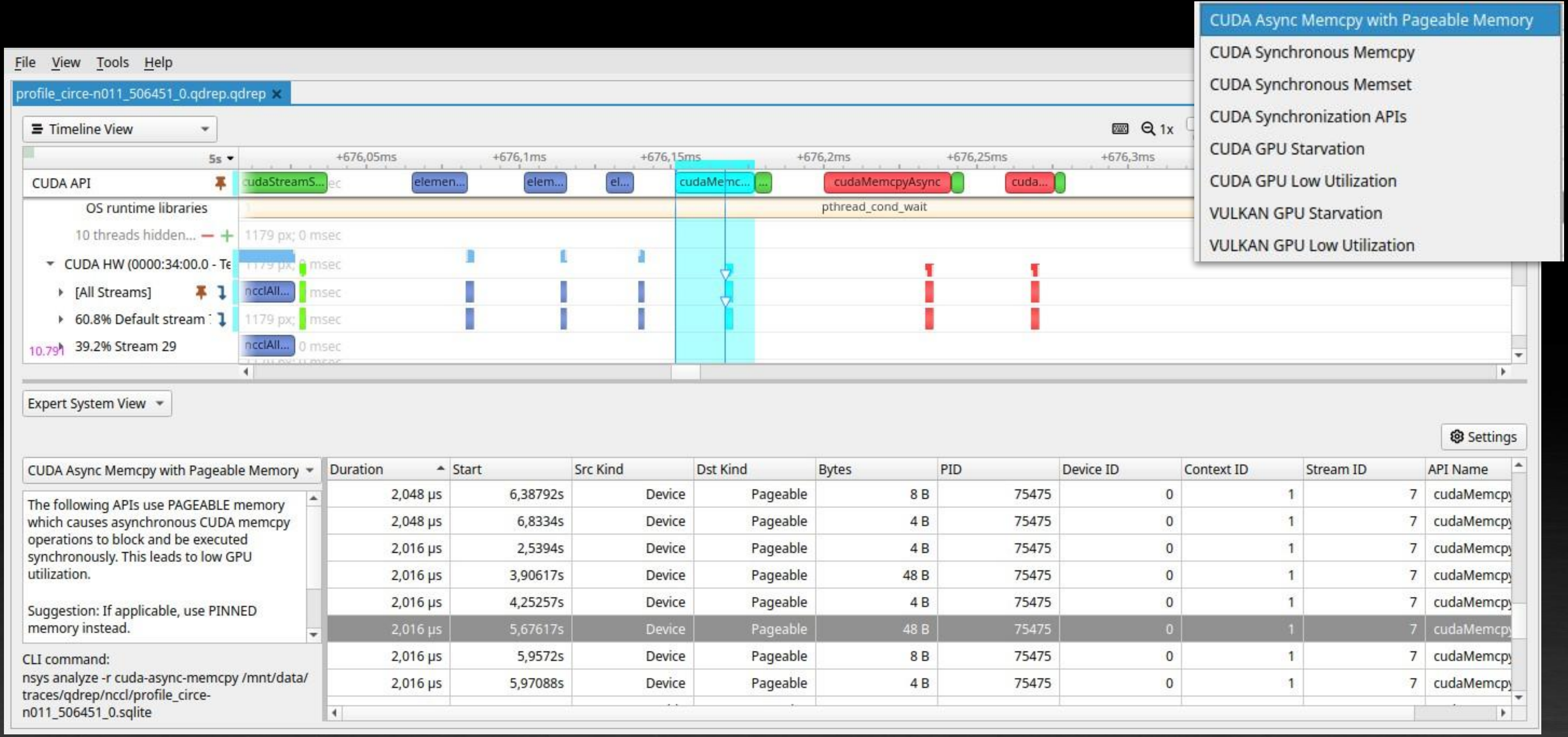
#	Name	Start	Duration	TID
149153	device_tea_leaf_ppcg_solve_calc_sd_new	11,4088s	3,265 µs	390019
149154	device_pack_bottom_buffer	11,4088s	3,247 µs	390019
149155	cudaMemcpy	11,4088s	1,838 ms	390019
149174	MPI_Isend	11,4106s	1,564 µs	390019
149175	MPI_Irecv	11,4106s	1,420 µs	390019
149176	MPI_Waitall	11,4106s	1,772 ms	390019
149199	cudaMemcpy	11,4124s	77,788 µs	390019
149201	device_unpack_bottom_buffer	11,4125s	5,232 µs	390019
149202	device_tea_leaf_ppcg_solve_update_r	11,4125s	3,334 µs	390019
149203	device_tea_leaf_ppcg_solve_calc_sd_new	11,4125s	3,268 µs	390019
149204	device_pack_bottom_buffer	11,4125s	3,031 µs	390019
149205	cudaMemcpy	11,4125s	1,853 ms	390019
149224	MPI_Isend	11,4143s	2,081 µs	390019
149225	MPI_Irecv	11,4143s	1,191 µs	390019
149226	MPI_Waitall	11,4143s	1,786 ms	390019
149227	libucp.so.0.0.0!ucp_worker_progress	11,4144s	-	390019
149228	libucp.so.0.0.0!ucp_worker_progress	11,4144s	-	390019
149229	libopen-pal.so.40.20.5!opal_progress	11,4145s	-	390019
149230	libuct.so.0.0.0!uct_mm_iface_progress	11,4146s	-	390019
149231	libopen-pal.so.40.20.5!opal_progress	11,4147s	-	390019

Description:

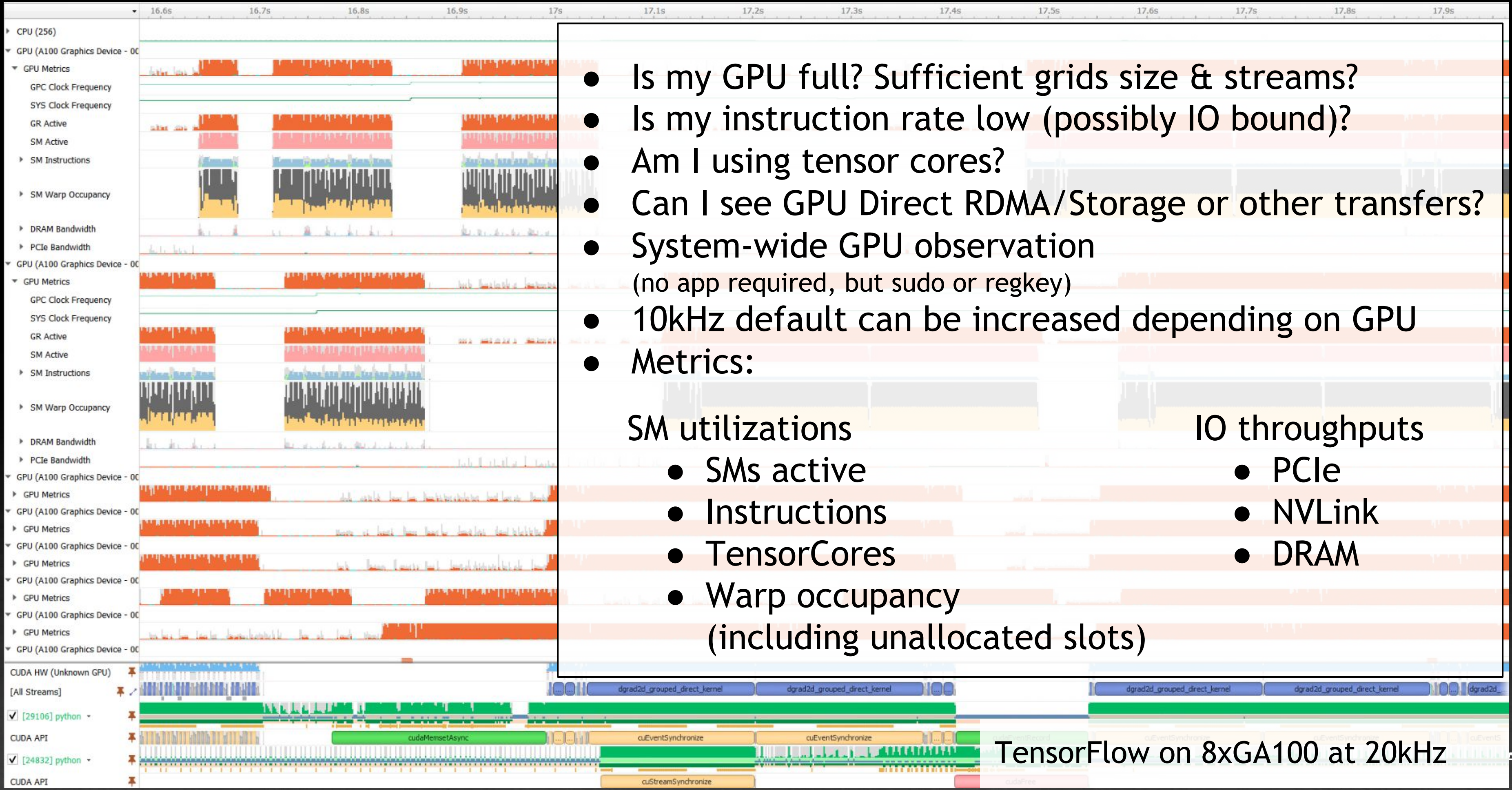
Call to cudaMemcpy
Memory copies
Begins: 11,4088s
Ends: 11,4106s (+1,838 ms)
Return value: 0
Correlation ID: 40146

Call stack:
libcuda.so.460.27.04**[12 Frames]**
libcuda.so.460.27.04!**cuMemcpyDtoH_v2**
libcudart.so.11.1.74**[2 Frames]**
libcudart.so.11.1.74!**cudaMemcpy**
tea_leaf!
TealeafCudaChunk::packUnpackAllBuffers(...)
tea_leaf!**cuda_pack_buffers_**
tea_leaf!**__tea_module_MOD_tea_exchange**
tea_leaf!
__update_halo_module_MOD_update_halo
tea_leaf!
__tea_leaf_ppcg_module_MOD_tea_leaf_run_ppcg_inner_steps
tea_leaf!**__tea_leaf_module_MOD_tea_leaf**
tea_leaf!**diffuse_**
[Unknown]!**0x0**
[Broken backtraces]!**[Broken backtraces]**

Expert System



GPU Metrics Sampling



- Is my GPU full? Sufficient grids size & streams?
- Is my instruction rate low (possibly IO bound)?
- Am I using tensor cores?
- Can I see GPU Direct RDMA/Storage or other transfers?
- System-wide GPU observation
  (no app required, but sudo or regkey)
- 10kHz default can be increased depending on GPU
- Metrics:

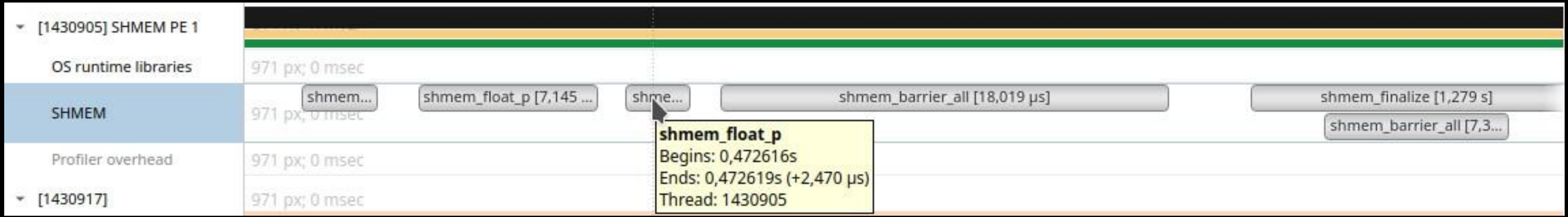
SM utilizations
- SMs active
- Instructions
- TensorCores
- Warp occupancy
  (including unallocated slots)

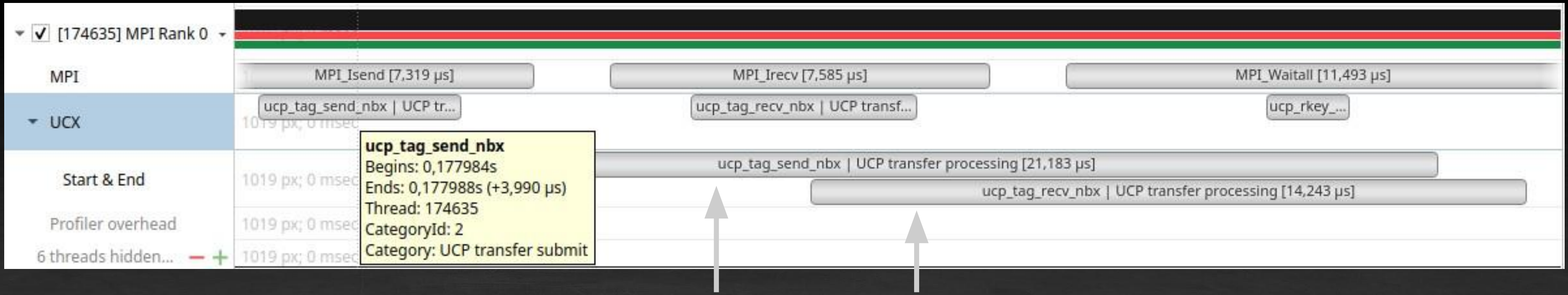
IO throughputs
- PCIe
- NVLink
- DRAM

TensorFlow on 8xGA100 at 20kHz

Communication Libraries & Multi-Node Support

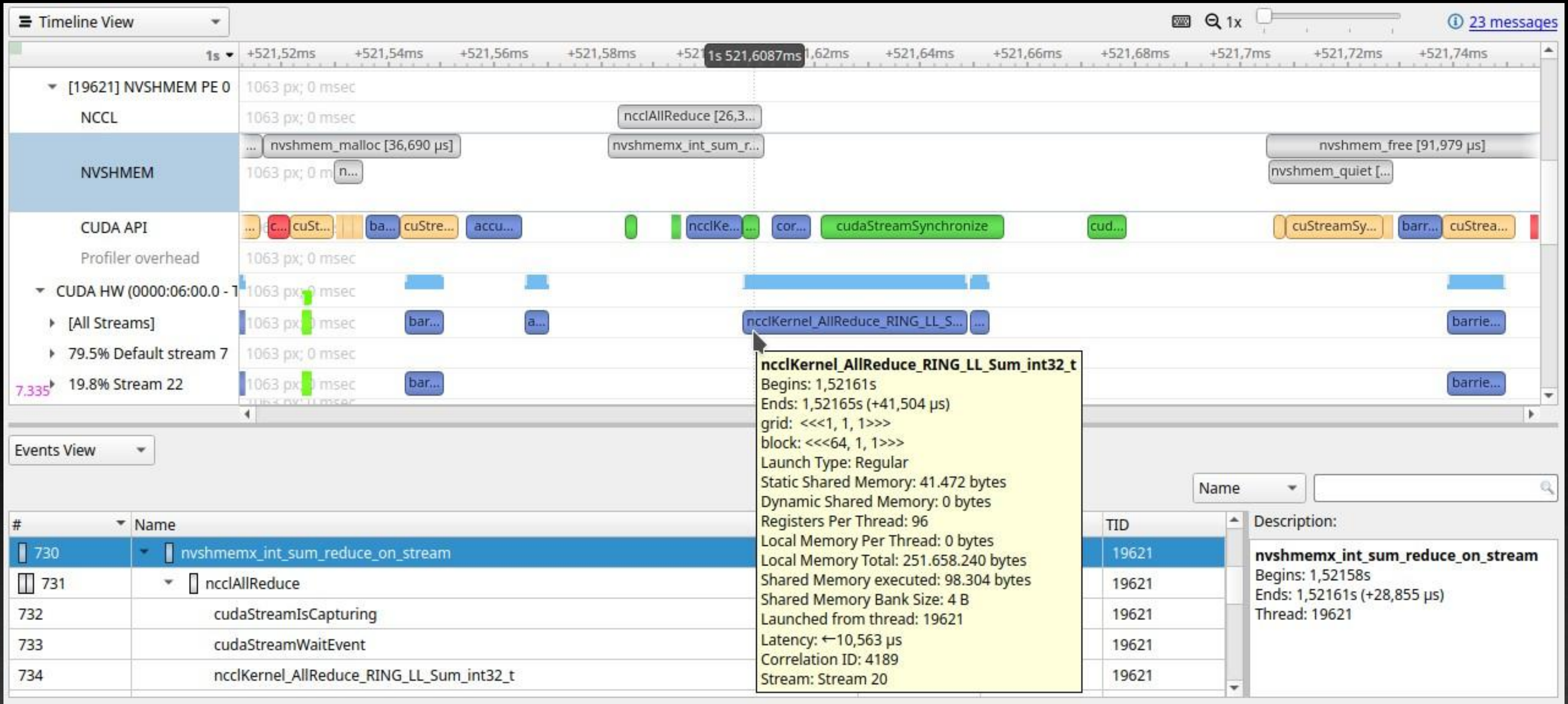
OpenSHMEM, MPI and UCX



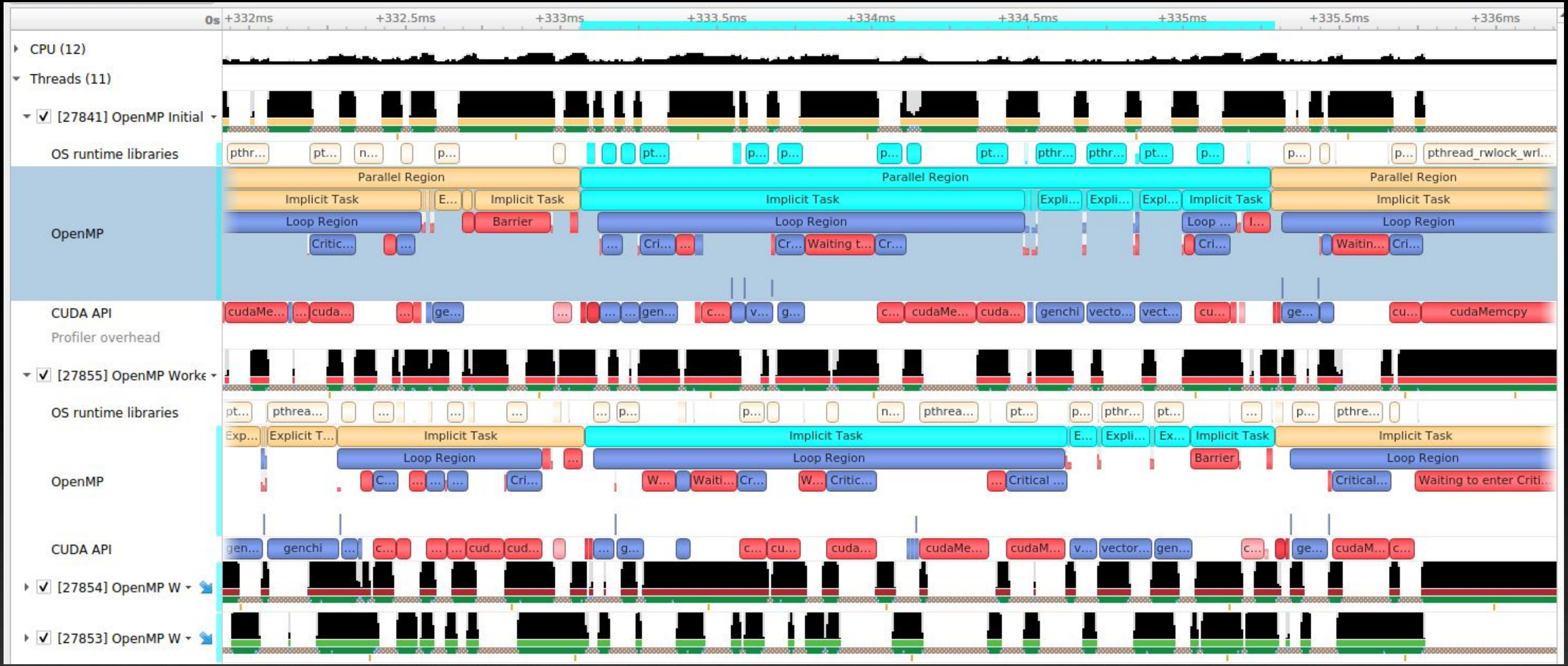


Completion tracking of non-blocking UCP communication operations

NVSHMEM and NCCL



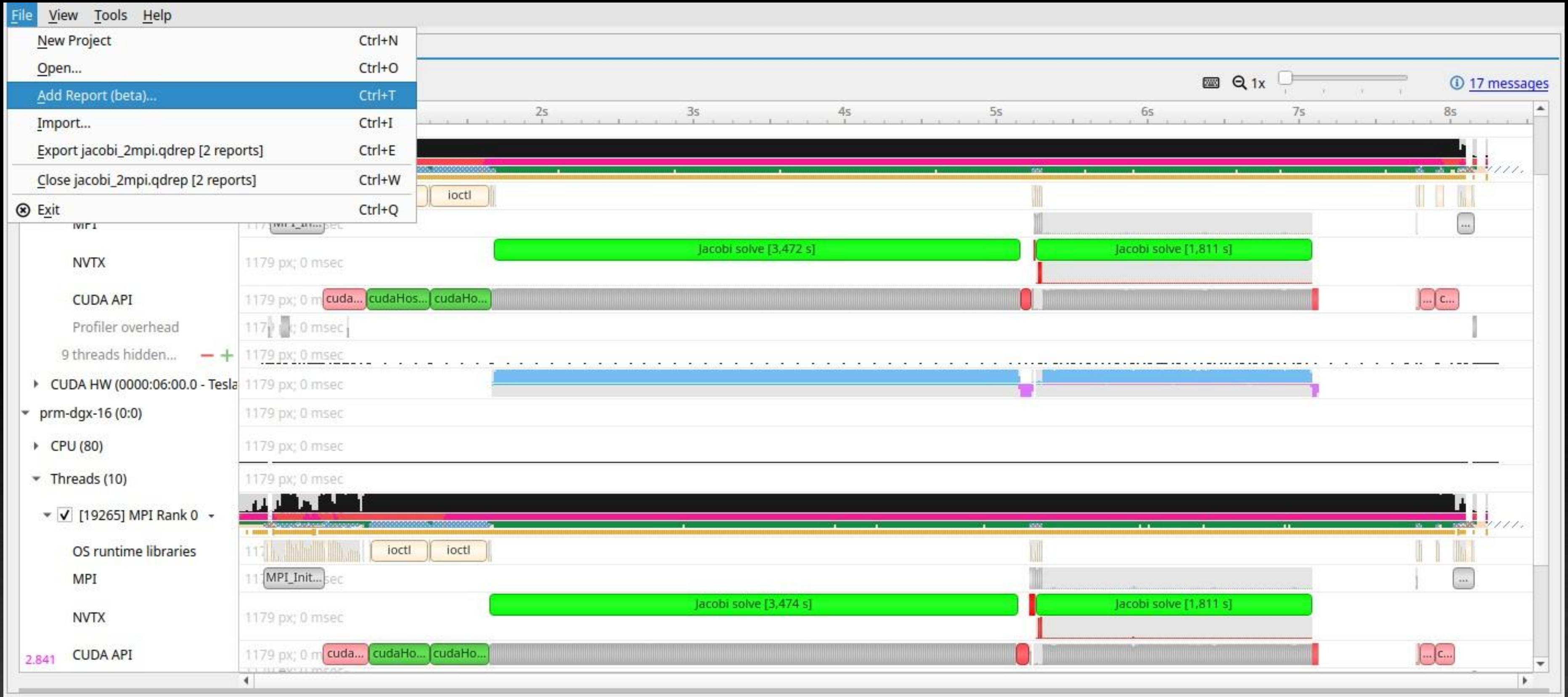
OpenMP - Correlation Highlighting



OMPT-capable OpenMP runtime is required.

NVIDIA

Loading Multiple Reports into one Timeline



Nsight Systems - Summary

Today

- Mainly used for single server profiling runs
- Focused on GPU workloads, API trace, thread samples
- Cluster profiling
  - Load multiple reports into one timeline
  - MPI (including communication parameters), OpenSHMEM, UCX tracing
  - NIC metrics sampling (congestions, read/write bandwidth)

Roadmap

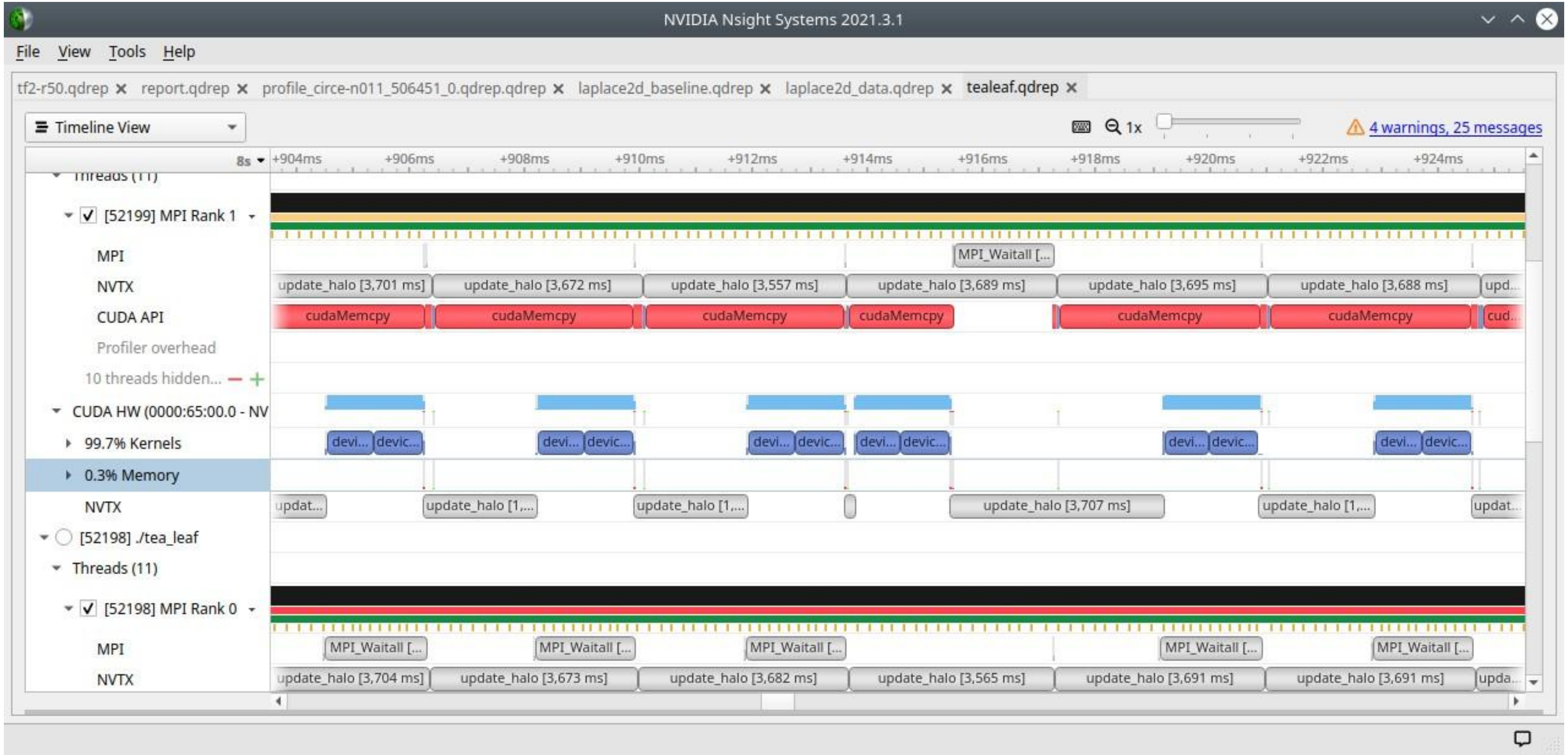
- Improve multi-node support, e.g. correlation across nodes, switch statistics, etc.
- Determine latency, congestion, hot devices (NICs, switches, network storage)

THANK YOU!

Download	https://developer.nvidia.com/nsight-systems NOTE: website version is newer than CUDA Toolkit version
Docs	https://docs.nvidia.com/nsight-systems/index.html
Forums	https://devtalk.nvidia.com
Email	nsight-systems@nvidia.com
Blogs	• https://developer.nvidia.com/blog/nvidia-nsight-systems-containers-cloud • https://developer.nvidia.com/blog/nsight-systems-exposes-gpu-optimization • https://developer.nvidia.com/blog/understanding-the-visualization-of-overhead-and-latency-in-nsight-systems • https://developer.nvidia.com/blog/nvidia-tools-extension-api-nvtx-annotation-tool-for-profiling-code-in-python-and-c-c/

Hands-on TeaLeaf

Hands-on TeaLeaf



Sample from https://github.com/UK-MAC/TeaLeaf

Build TeaLeaf

```
# Get the source code
git clone https://github.com/UK-MAC/TeaLeaf_CUDA.git ; cd TeaLeaf_CUDA

# Get Fortran NVTX module and apply provided TeaLeaf NVTX patch
wget
https://raw.githubusercontent.com/maxcuda/NVTX_example/master/nvtx.f90
git apply /p/project/training2123/work/dietrich3/tea_nvtx.diff

# Load the environment modules
module add Stages/2022 NVHPC OpenMPI CUDA Nsight-Systems
#source /p/project/training2123/work/schmitt5/load_modules.sh

# Build TeaLeaf
# if you did not apply the patch modify the Makefile:
#   add AMPERE architecture and -std=c++14 flag for nvcc
make -e COMPILER=PGI
```

Alternative: use the pre-compiled binary with minimal NVTX instrumentation

/p/project/training2123/work/dietrich3/TeaLeaf_CUDA/tea_leaf

Profile TeaLeaf

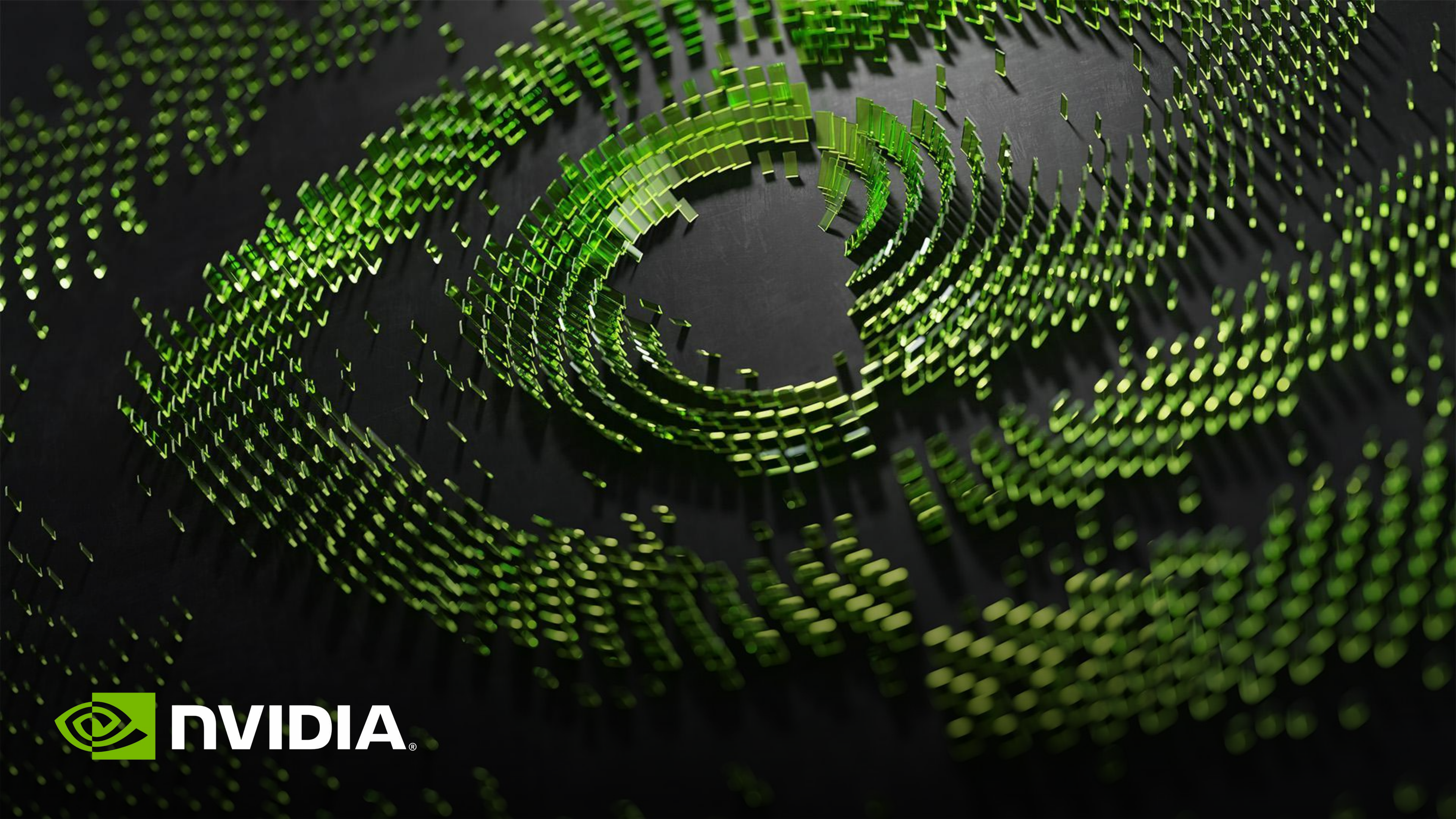
```
# Run TeaLeaf without profiling
srun -A training2123 -p booster -N 1 --ntasks-per-node=2 --gres=gpu:2 -t 10
./tea_leaf

# Profile with NVTX
srun -A training2123 -p booster -N 1 --ntasks-per-node=2 --gres=gpu:2 -t 10
\ nsys profile -t cuda,mpi,ucx,osrt,nvtx --mpi-impl=openmpi \
-y 1 -d 8 --kill=none -o tea-nvtx.%q{SLURM_PROCID} --stats=true ./tea_leaf


# Profile with CPU and GPU sampling
srun -A training2123 -p booster -N 1 --ntasks-per-node=2 --gres=gpu:2 -t 10
\ nsys profile -t cuda,mpi,ucx,osrt,nvtx --mpi-impl=openmpi \
-y 1 -d 8 --kill=none -o tea-sampling%q{SLURM_PROCID} \
--backtrace=dwarf --sampling-period=3000000 \
--gpu-metrics-set=ga100 --gpu-metrics-device=0,1 \
--gpu-metrics-frequency=15000 \
./tea_leaf
```

Analyze the Profile

Investigate CLI stats output (`--stats`)

Open the report file in the Nsight Systems GUI

1. Start JupyterLab
2. Open Xpra Desktop
3. `module load Nsight-Systems`
4. `nsys-ui REPORTFILE.nsys-rep`
5. Add report of second MPI rank into timeline (via GUI: File -> Add report (beta)


NVIDIA