GPU PERFORMANCE ANALYSIS SCORE-P AND NVIDIA TOOLS

APRIL 8, 2022 I MICHAEL KNOBLOCH

GPU Analysis using the

SCORE-P ECOSYSTEM

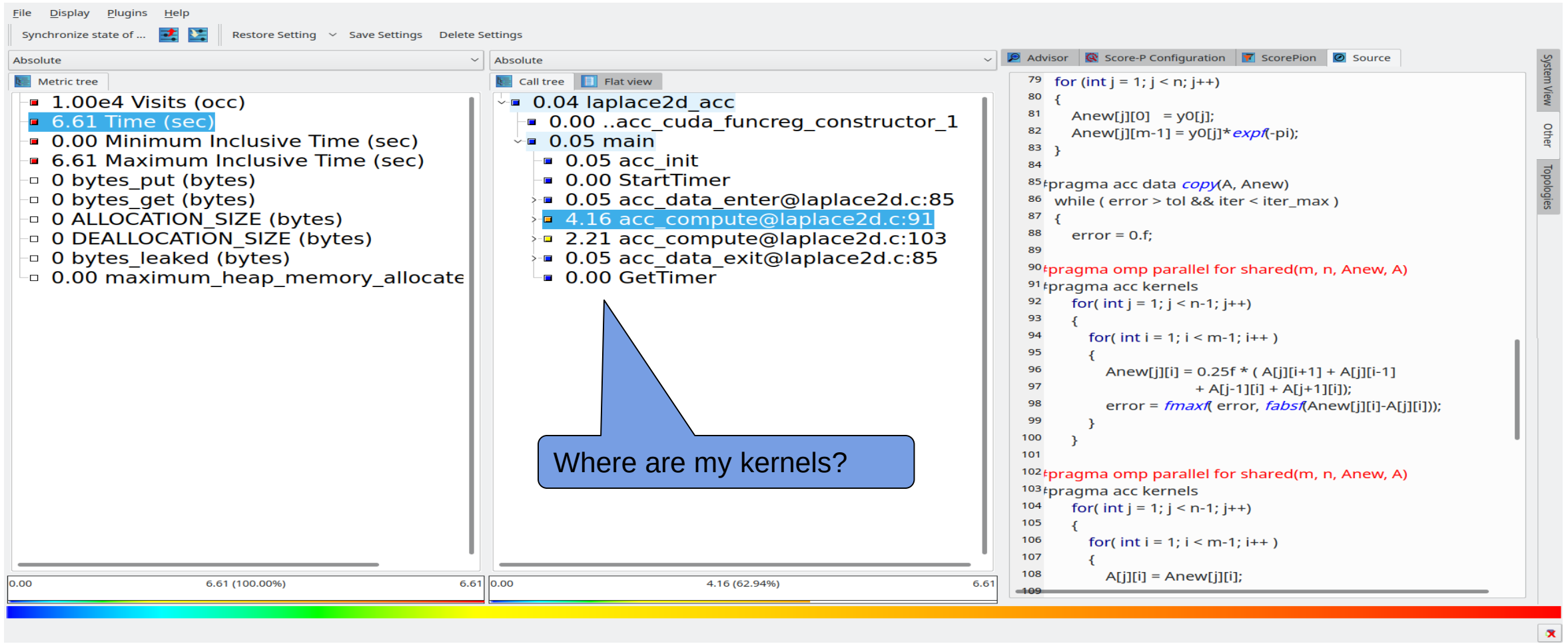
JÜLICH Forschungszentrum
JÜLICH SUPERCOMPUTING CENTRE

SCORE-P GPU MEASUREMENTS

- OpenACC
  - Prefix compiler and linker command with scorep --openacc
  - export ACC_PROFLIB=$SCOREP_ROOT/lib/libscorep_adapter_openacc_event.so
  - export SCOREP_OPENACC_ENABLE=yes
  - yes refers to: regions, wait, enqueue
  - Full list of options in User Guide
- CUDA
  - Prefix compiler and linker command with scorep --cuda
  - export SCOREP_CUDA_ENABLE=yes
  - yes refers to: runtime, kernel, memcpy
  - Full list of options in User Guide

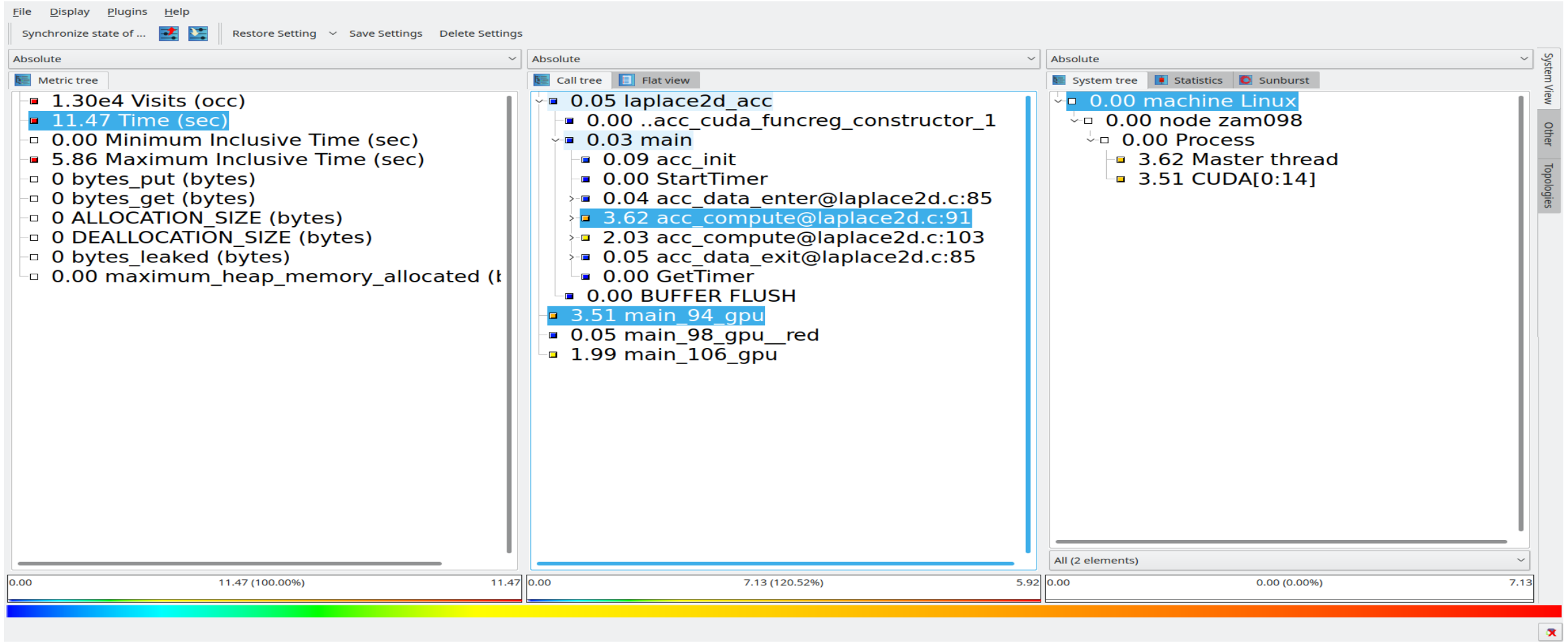
- OpenCL similar (use SCOREP_OPENCL_ENABLE=yes)

EXAMPLE: OPENACC



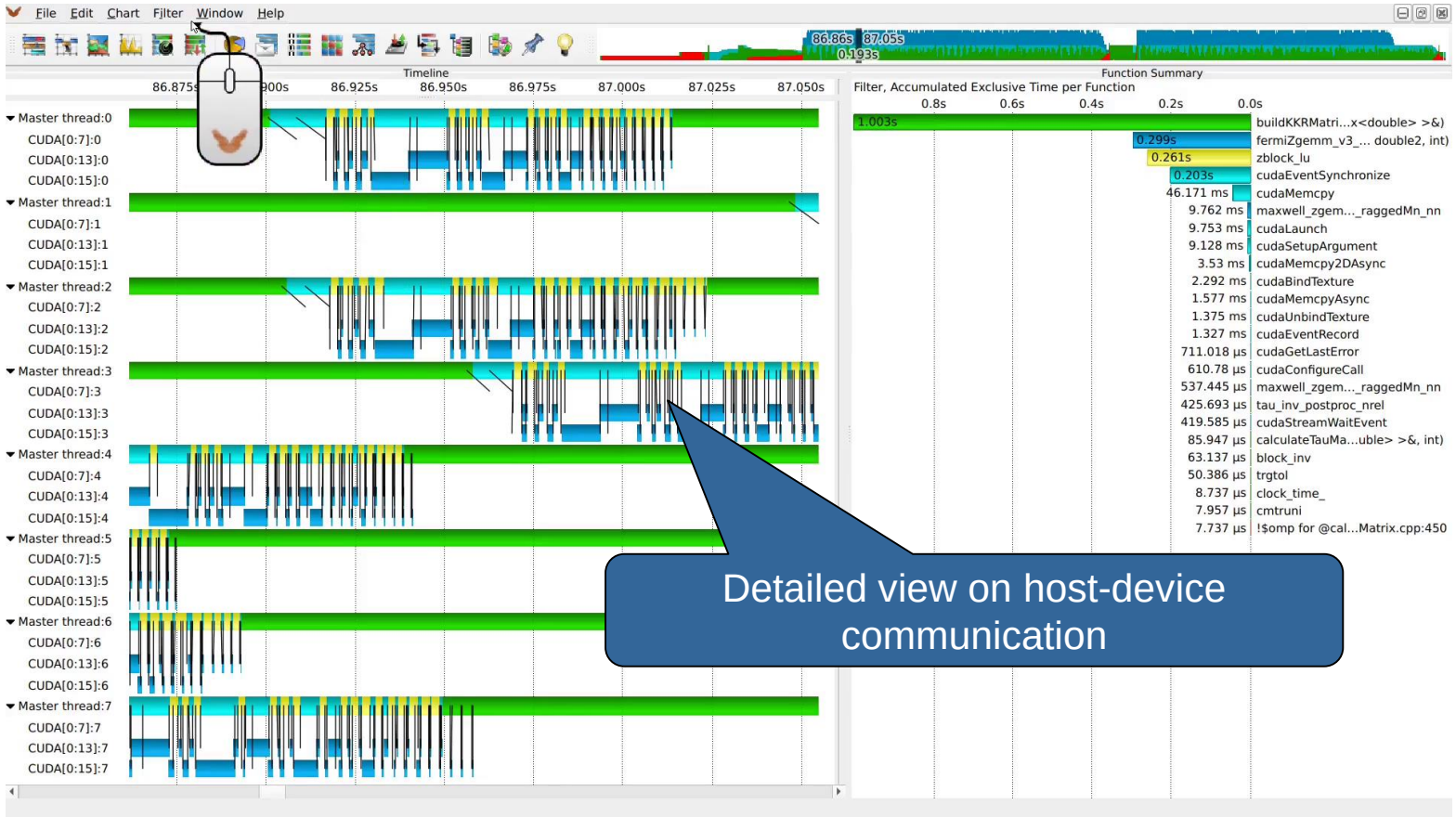
Pure OpenACC measurements contain host-side events only

EXAMPLE: OPENACC + CUDA



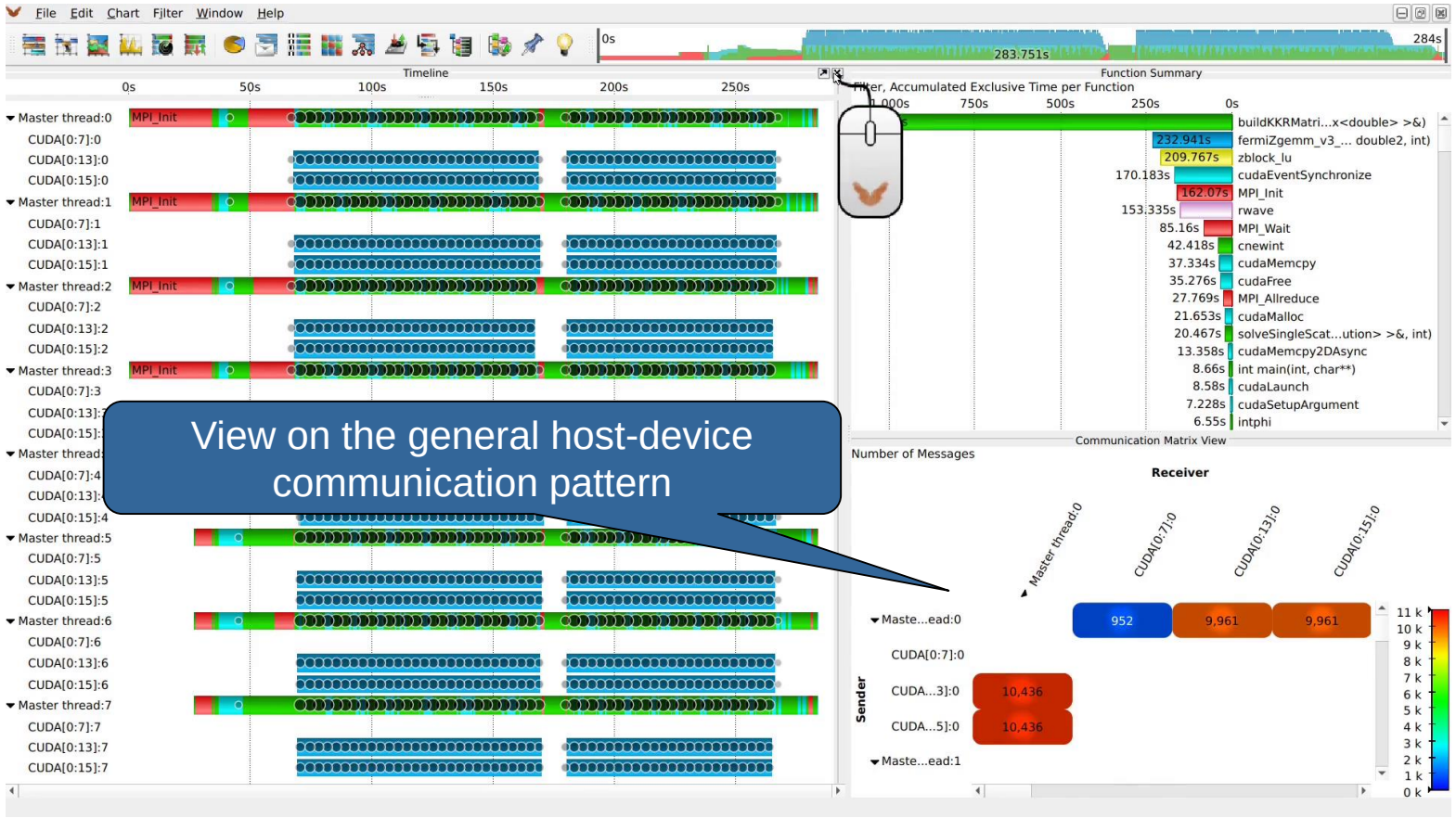
Enabling CUDA also shows kernels on the GPU

VAMPIR CUDA EXAMPLE



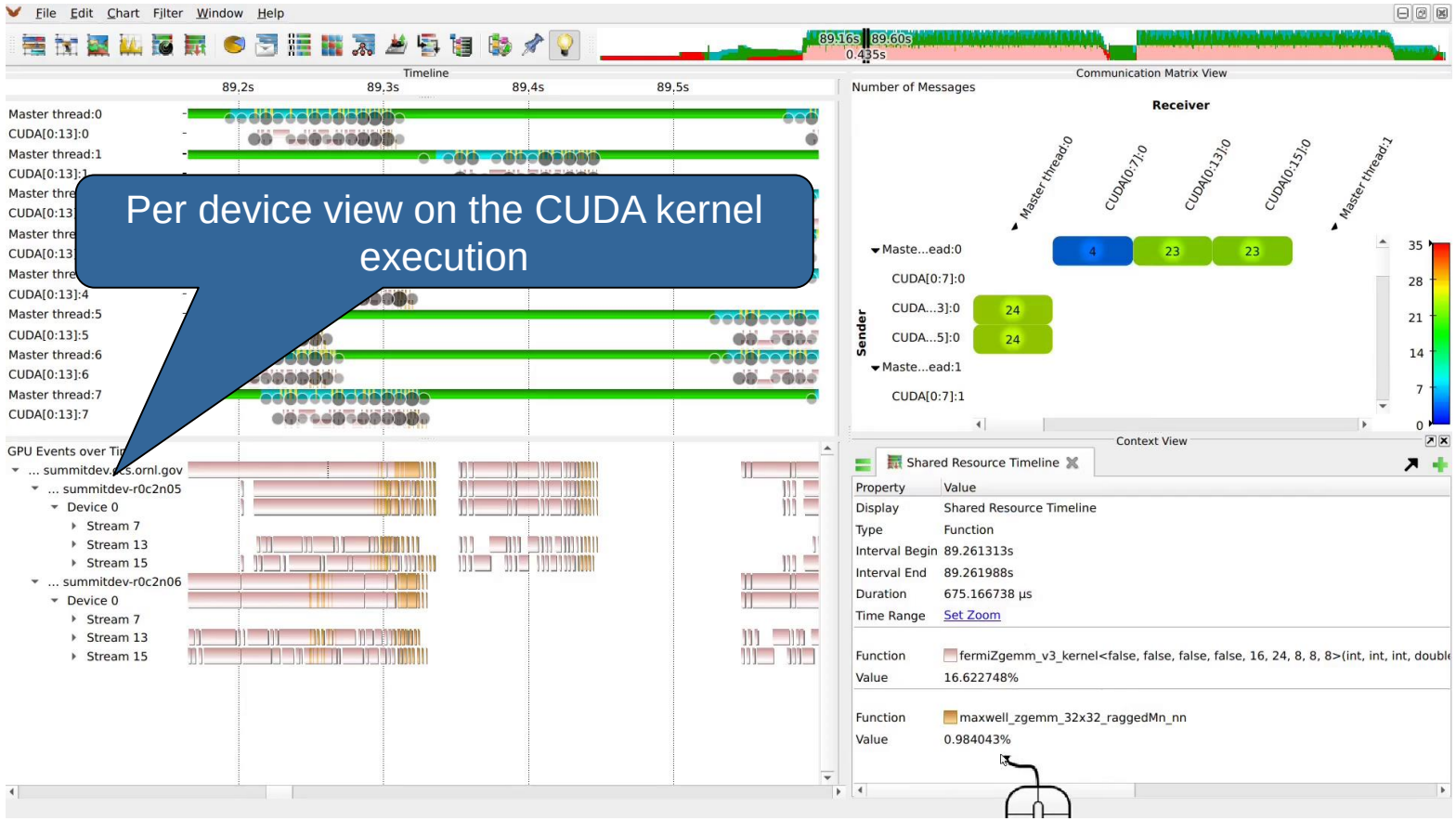
- Material science code LSMS
- CUDA is utilized for heavy computations
- CUDA streams are child's of the owning Process
- Allows an in-depth analysis of host-device communication

VAMPIR CUDA EXAMPLE



- Communication Matrix best for analyzing the general communication pattern
- Expectation: balanced communication, represented by a symmetric matrix
- Problem: communication with stream 7 is different

VAMPIR CUDA EXAMPLE



- Shared Resource Timeline offers a per device view on kernel executions
- Best suited for analyzing multi GPU per node scenarios
- Allows a dedicated analysis of kernel execution patterns
- Yields insights of the actual hardware usage

WHAT ABOUT SCALASCA AND GPUS?

- The Scalasca trace analyzer aborts if it encounters non-CPU locations, e.g. GPU streams
- However, GPU events are not too important for the analysis performed by Scalasca
- To enable a Scalasca analysis of a GPU application, either:
  - set SCOREP_CUDA_ENABLE=0 to disable all GPU events
  - set SCOREP_CUDA_ENABLE=runtime to get host-side events from the runtime

Attention:

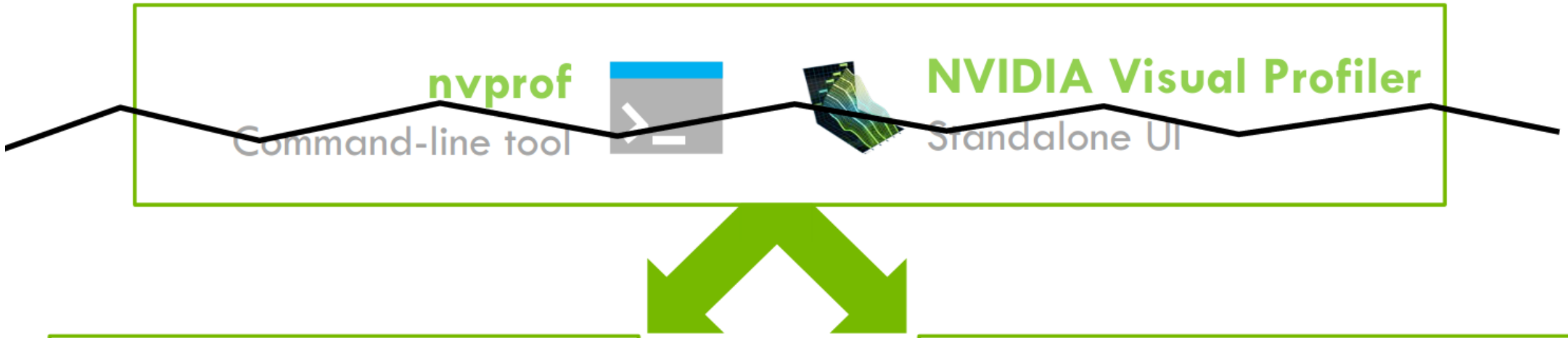
- Due to the asynchronous execution of GPU kernels, some advanced Scalasca analysis can be misleading, e.g. Critical-path analysis
- Host side might be idle, while GPU can still be busy

GPU Analysis using

NVIDIA NSIGHT

JÜLICH Forschungszentrum
JÜLICH SUPERCOMPUTING CENTRE

LEGACY TRANSITION



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

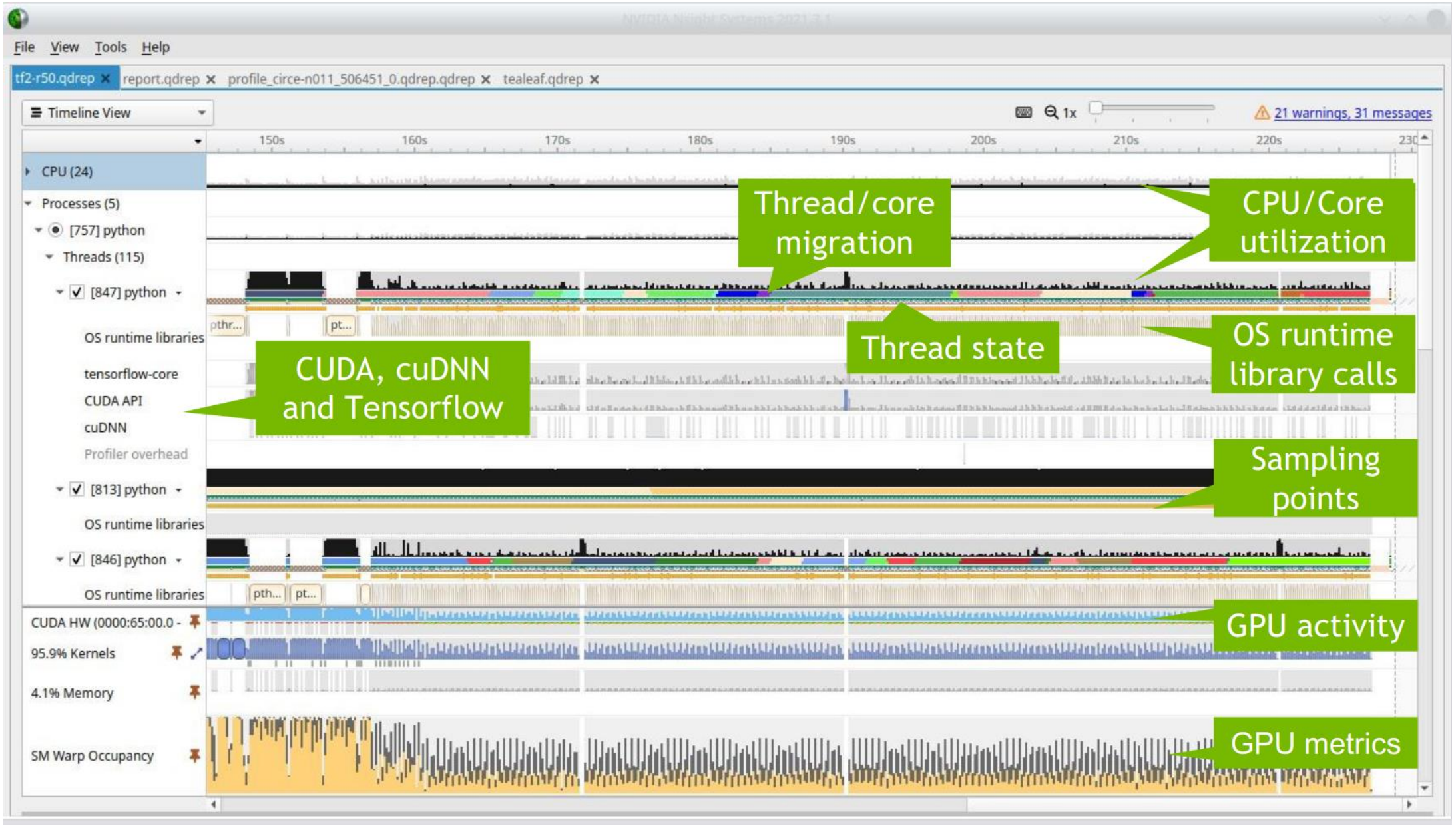
Very high freq GPU perf counters

Compare results (diff)

Incredible statistics & customizable

NSIGHT SYSTEM

- System-wide application tuning
- Locate optimization opportunities
  - Visualize millions of events on a timeline
  - See gaps of unused CPU and GPU time
- Balance workloads 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
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

USAGE OF CLI

- Basic profiling session: `nsys profile ./app`
- Interactive session (scriptable): `nsys start|launch|stop|cancel`
- Useful flags:
  - API tracing: `-t, --trace={cuda,nvtx,mpi,openacc,openmp,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//pattern for hostname, PID, etc//`
  - Callstack sampling:
  - CUDA memory usage: `--cuda-memory-usage=[true|`**`false`**`]`

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
```

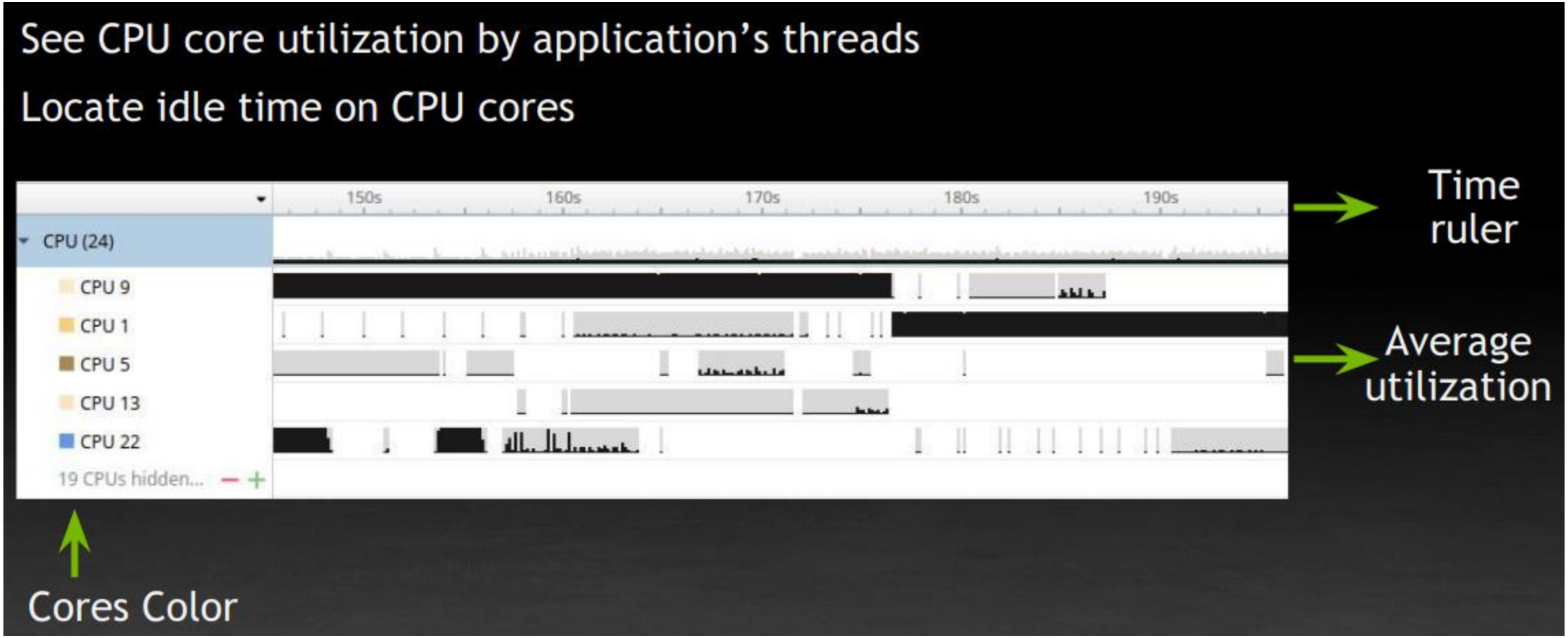
Set output report name with **`-o report_name_%q{OMPI_COMM_WORLD_RANK}`**
(for OpenMPI, **`PMI_RANK`** for MPICH and **`SLURM_PROCID`** for Slurm)

⇨ This creates one report file per MPI rank.

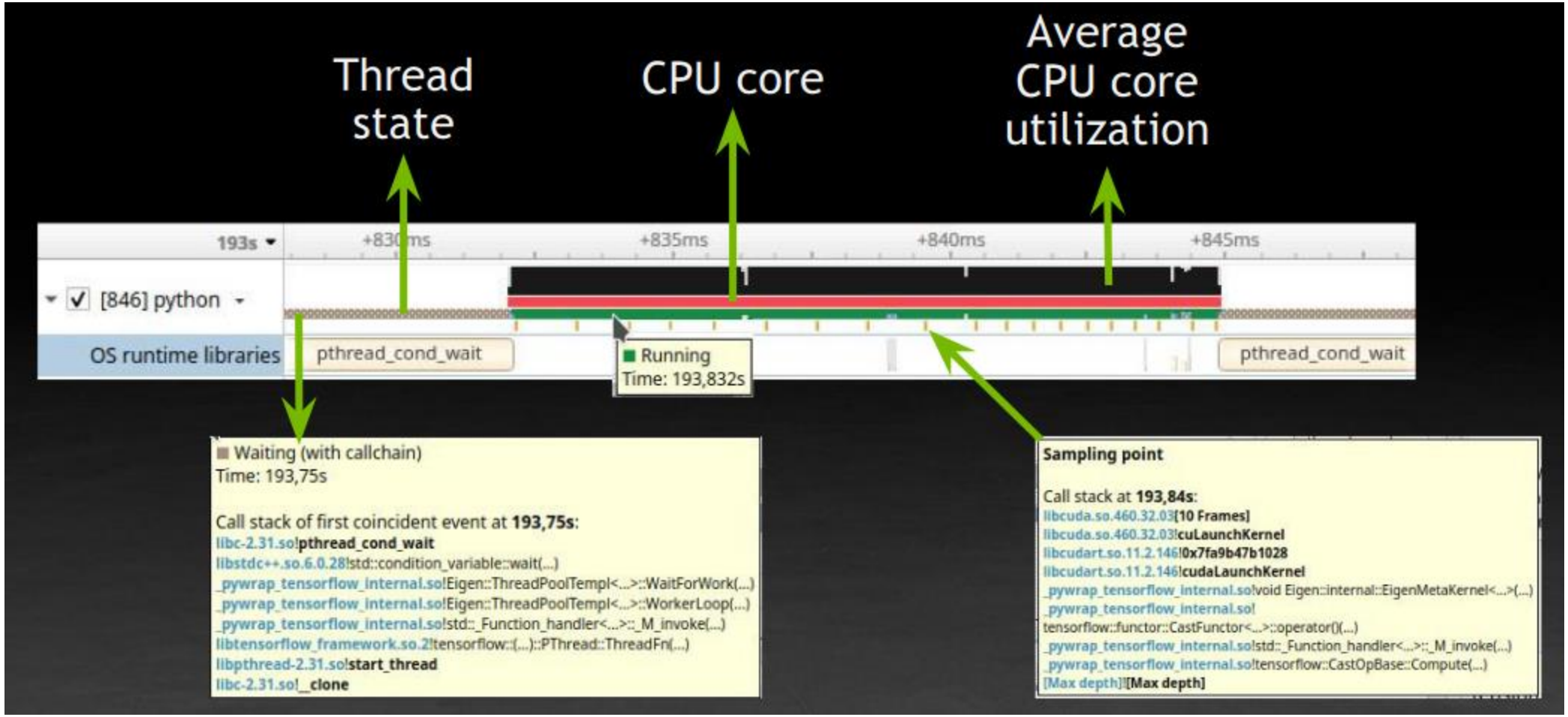
Profile only specific ranks. ⇨

```
#!/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
```

CPU CORES WORKLOAD

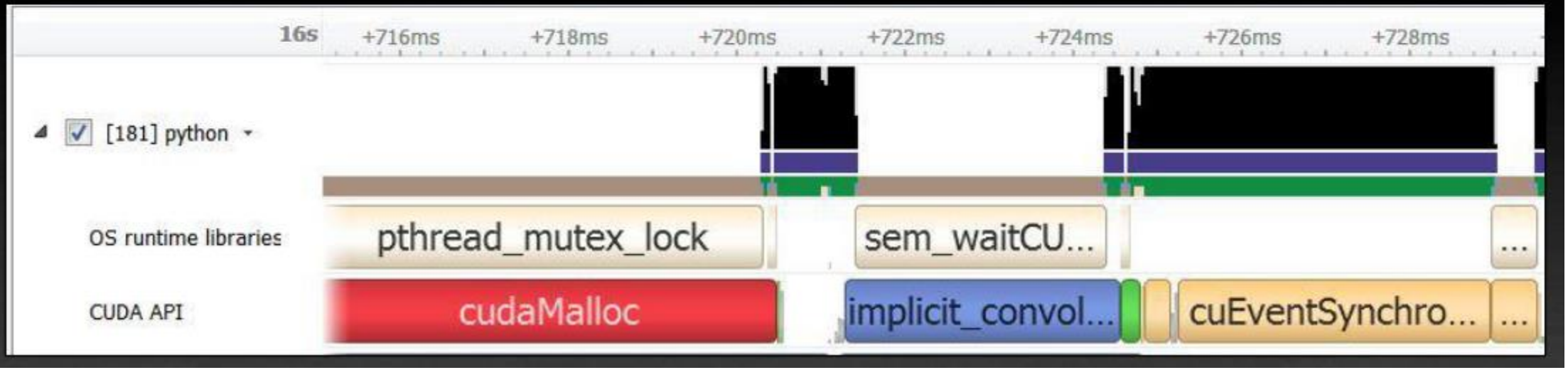


CPU THREAD ACTIVITY

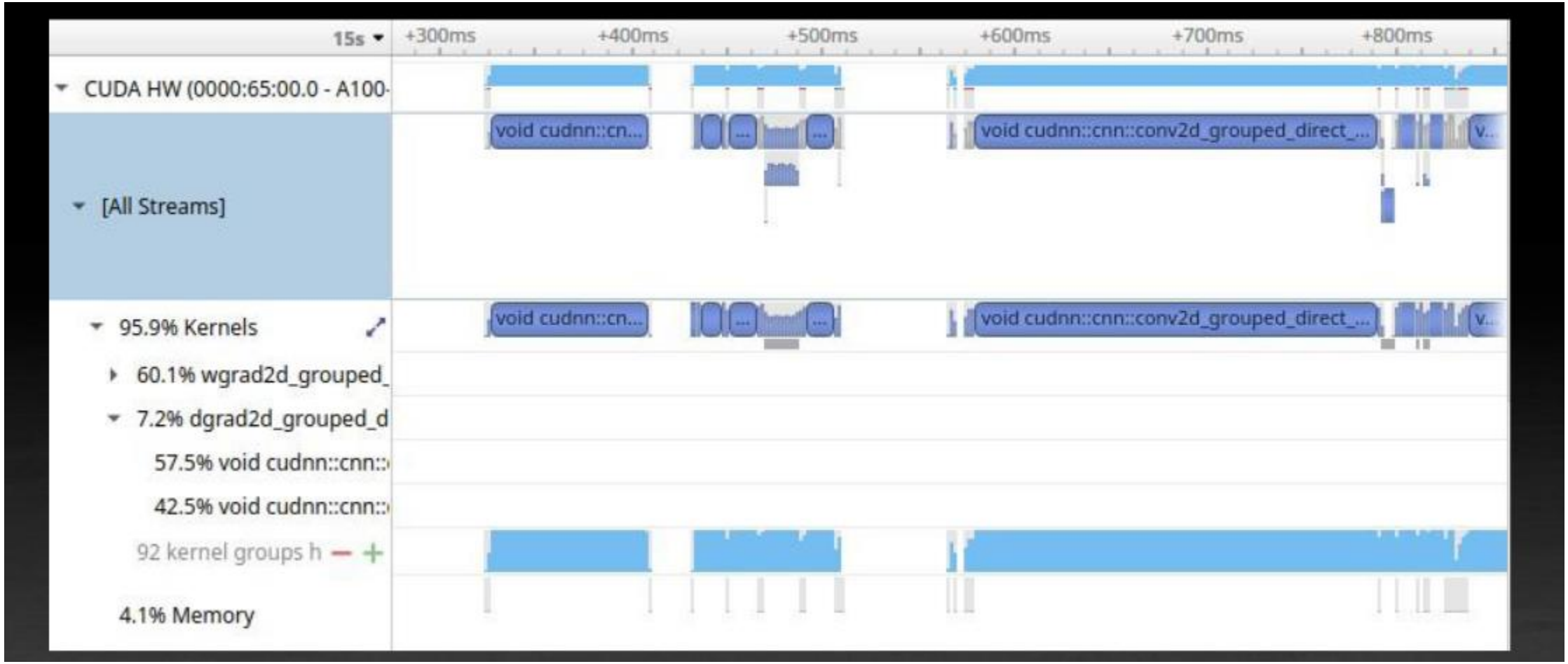


CUDA API

- 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

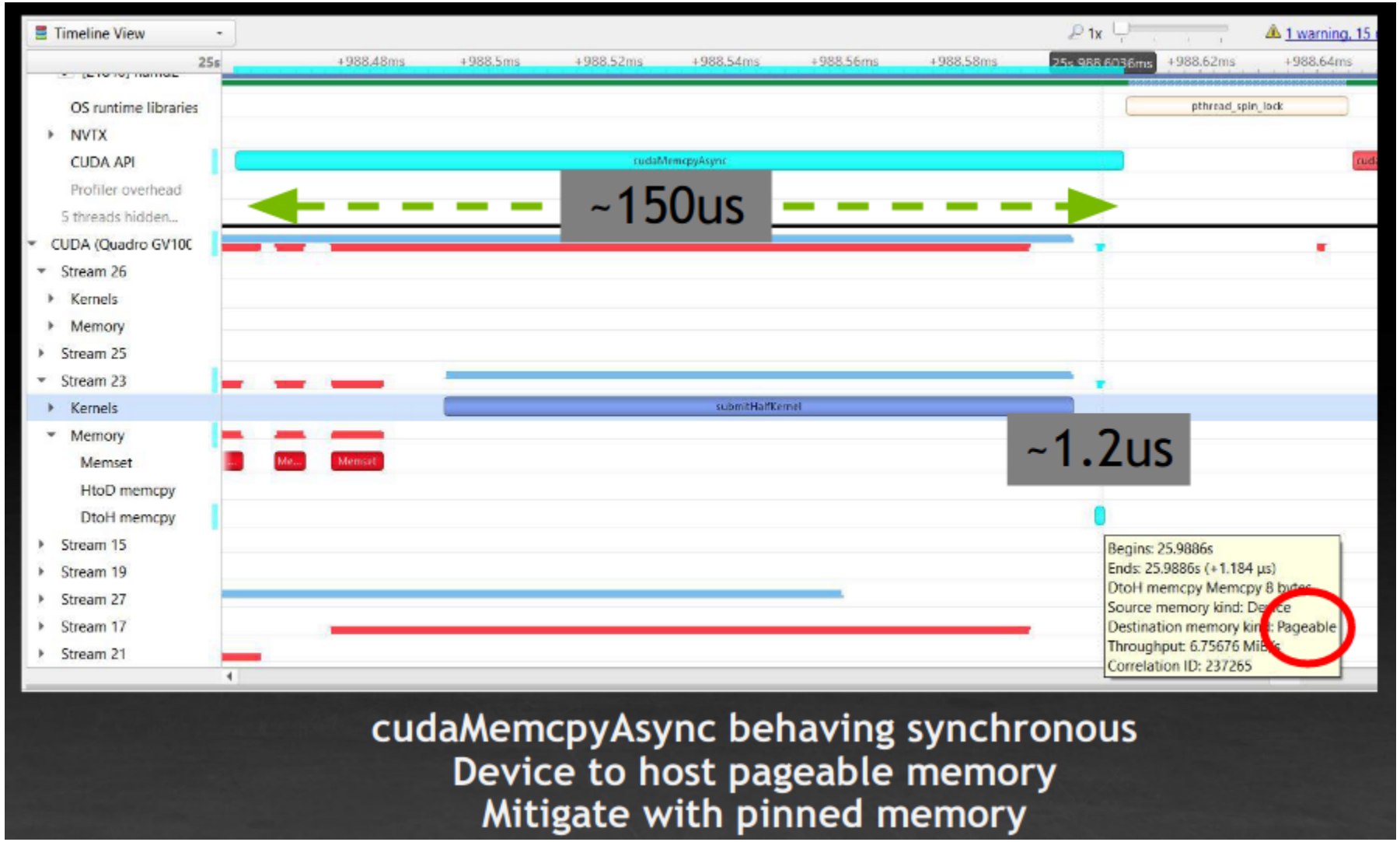


GPU UTILIZATION

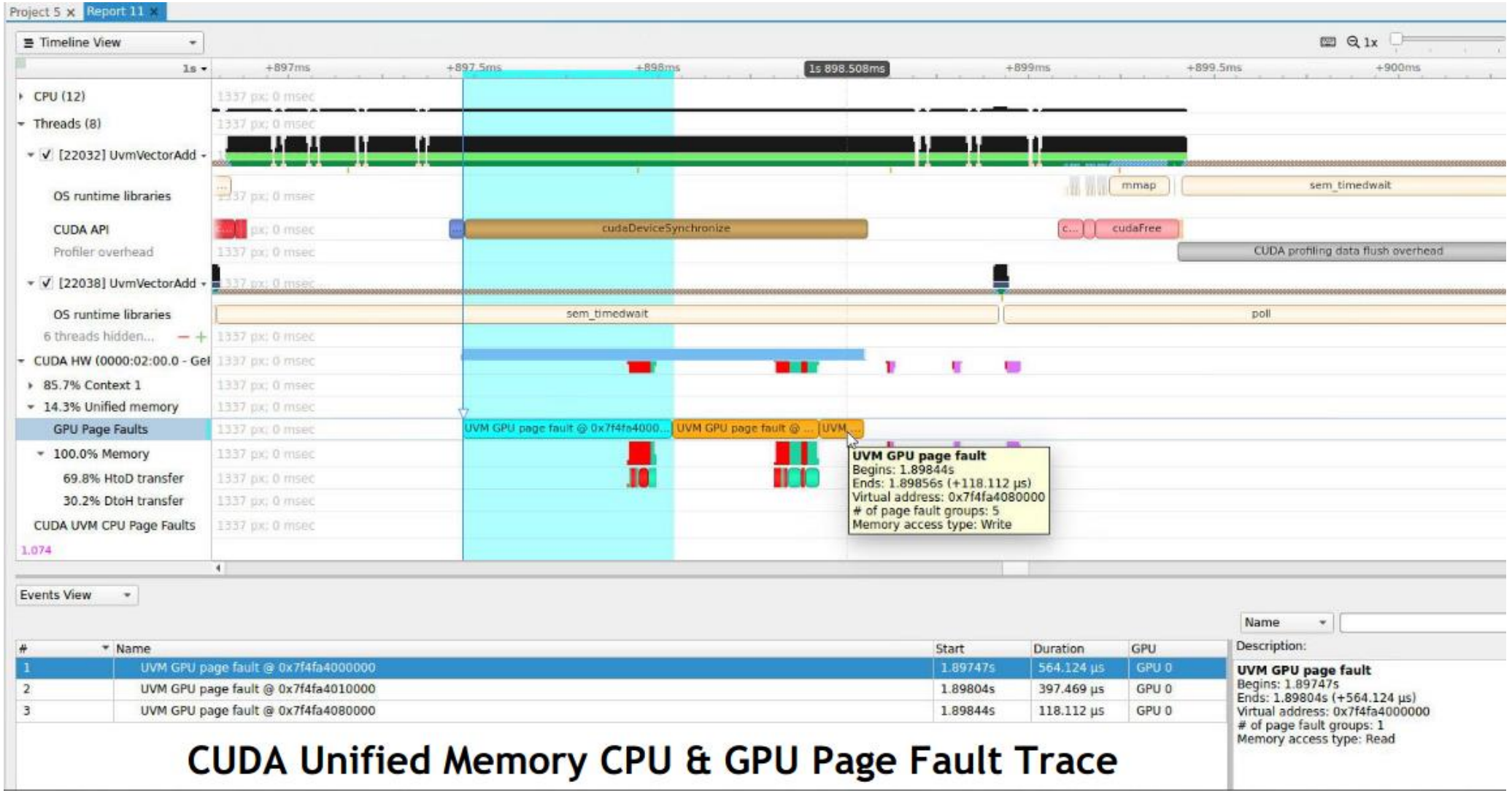


GPU UTILIZATION IN DETAIL

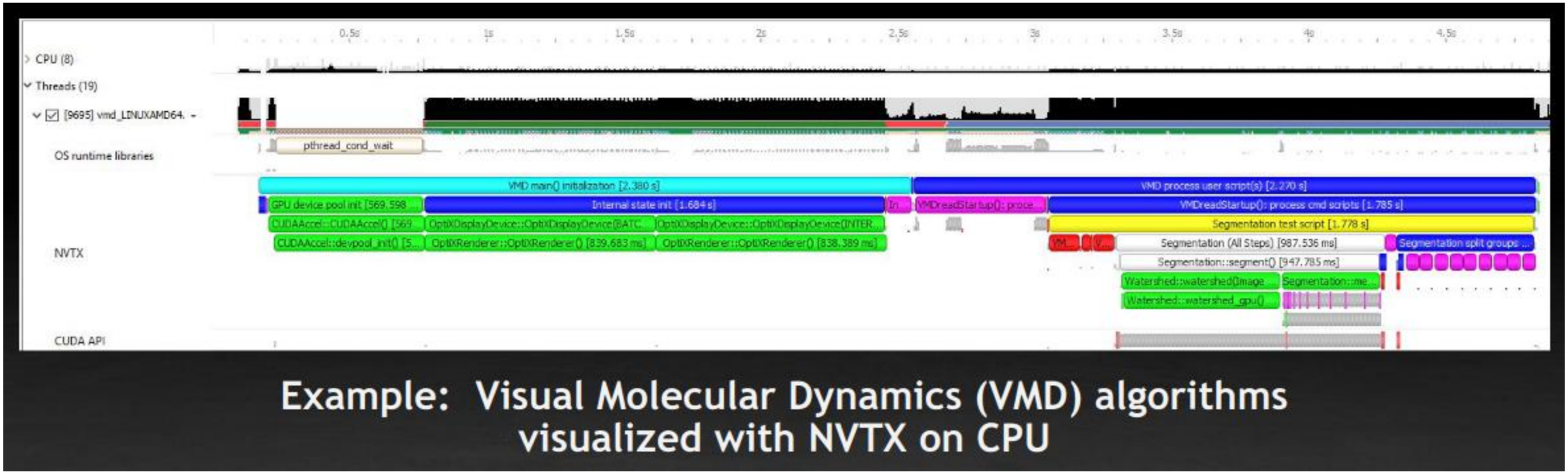
CUDA MEMORY TRANSFERS



UNIFIED MEMORY ANALYSIS

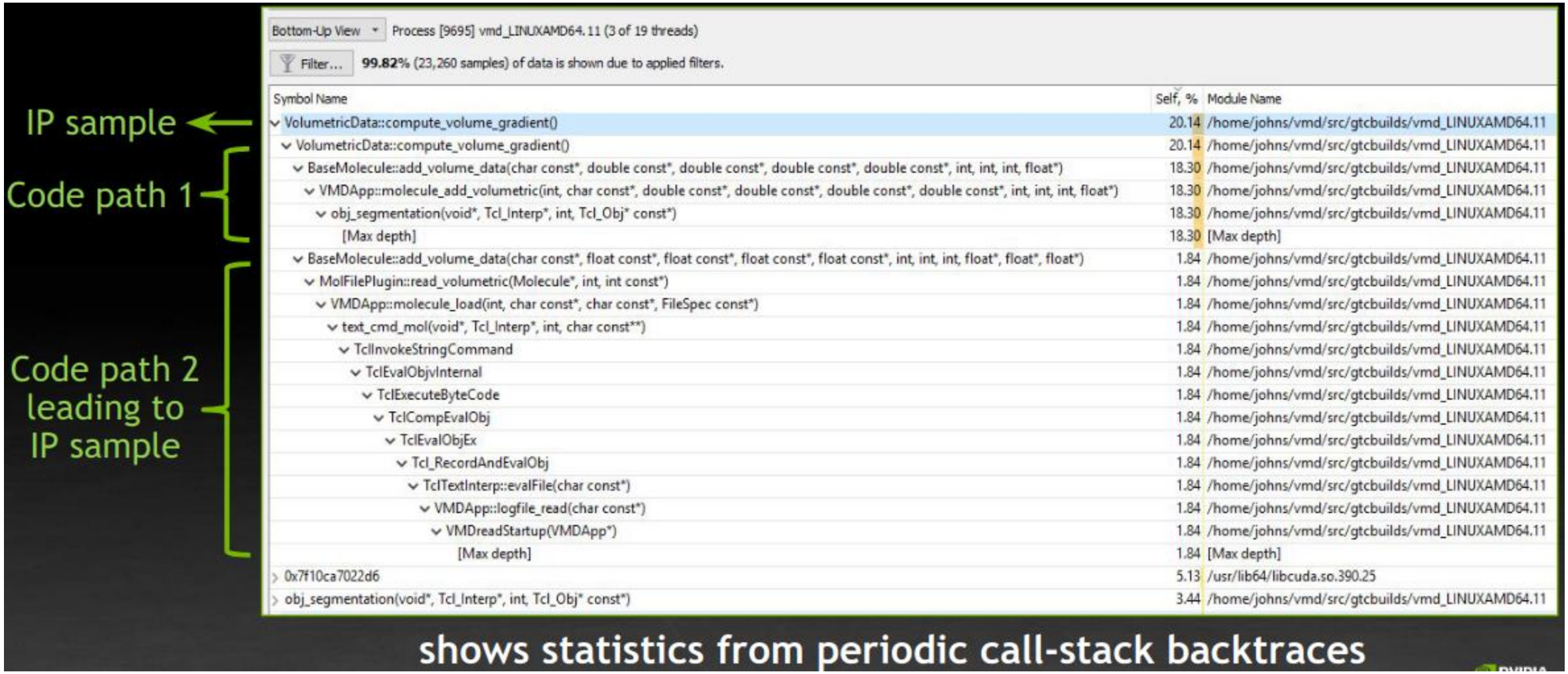


USER ANNOTATIONS WITH NVTX

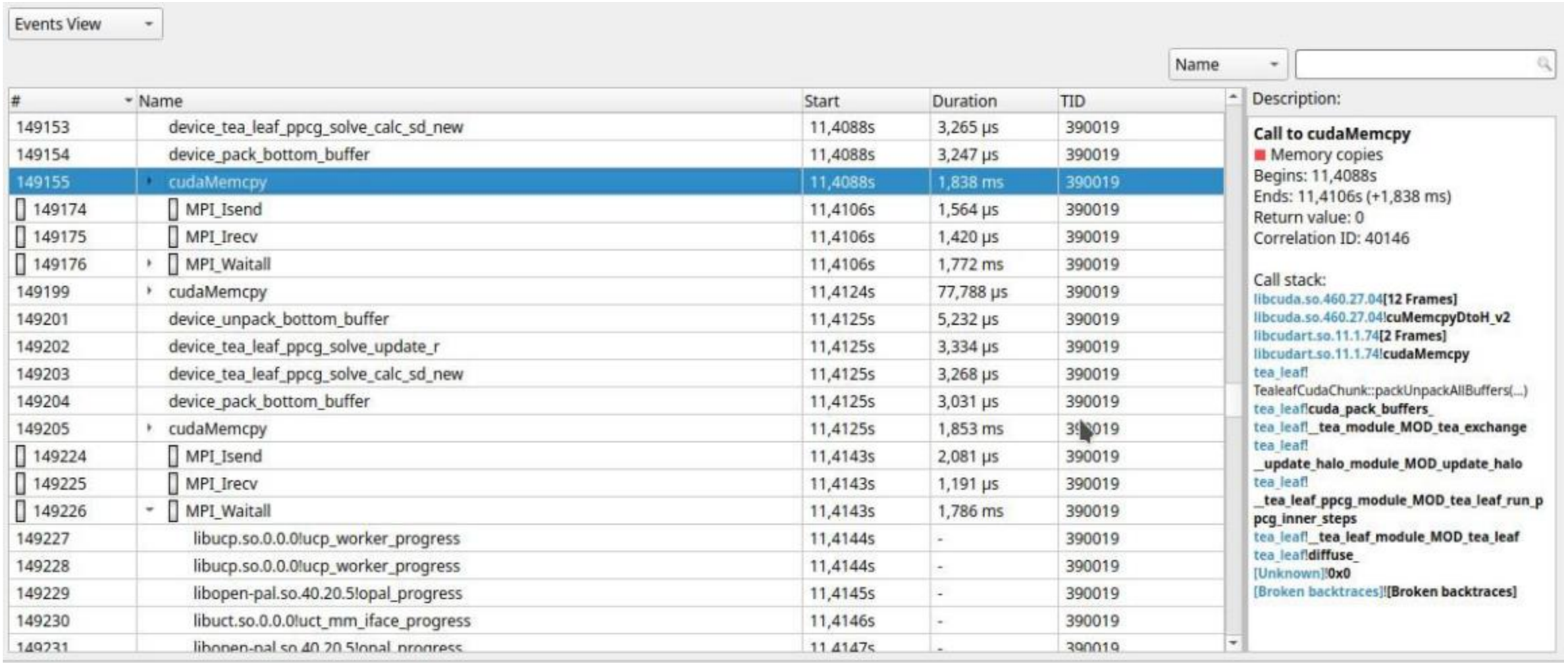


Example: Visual Molecular Dynamics (VMD) algorithms visualized with NVTX on CPU

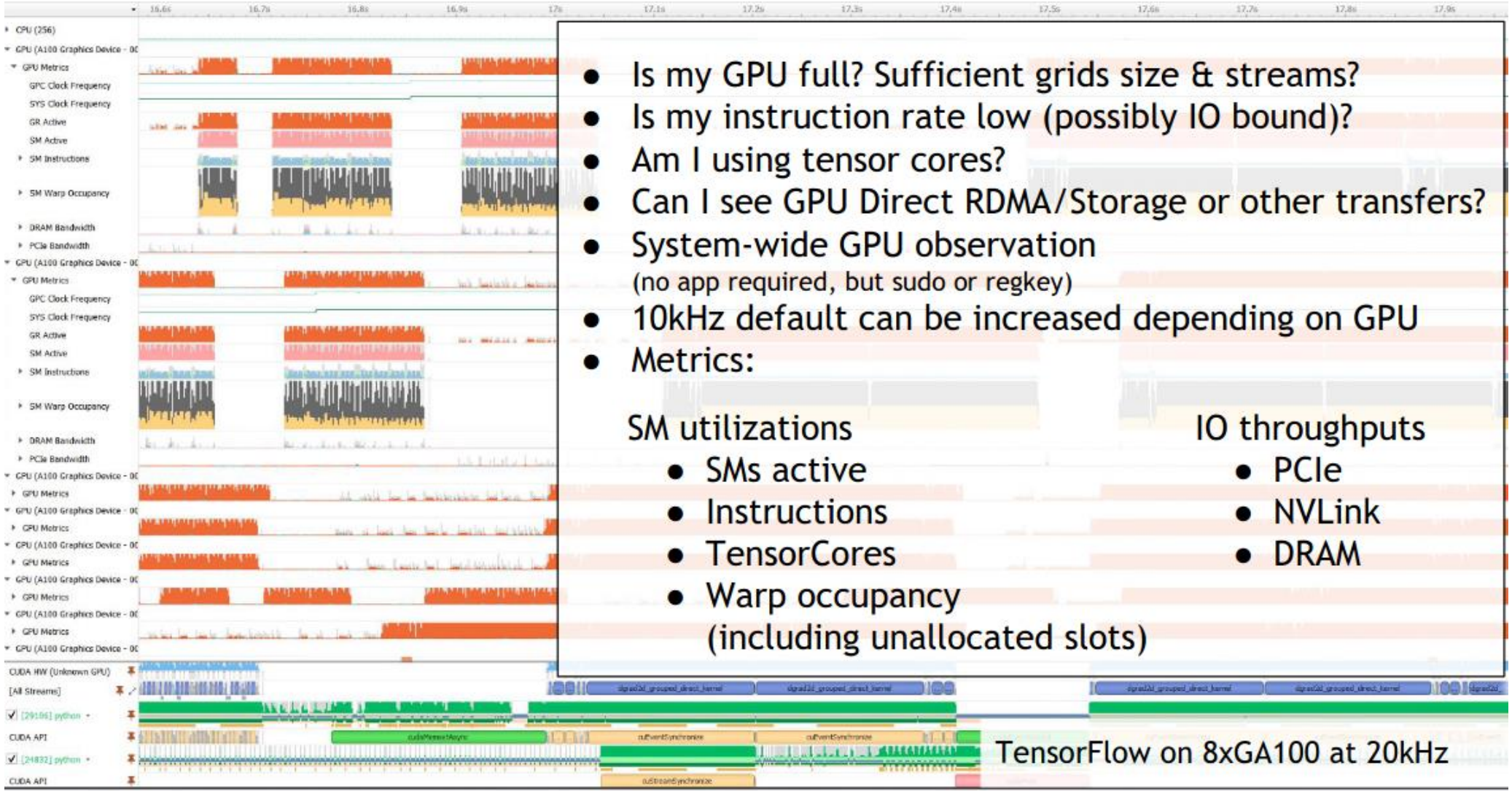
STATISTICAL SAMPLING



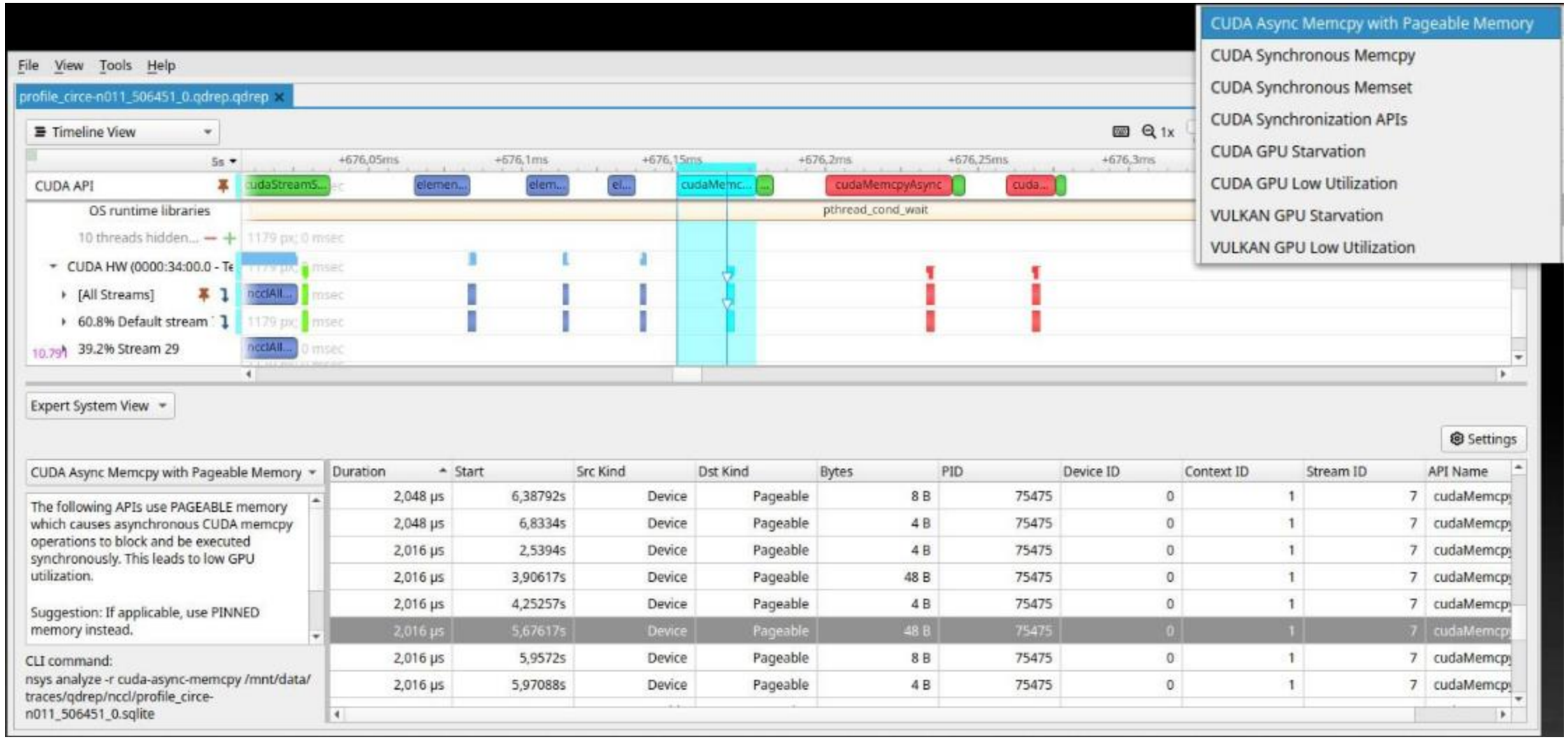
EVENTS VIEW


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_p
pcg_inner_steps
tea_leaf!__tea_leaf_module_MOD_tea_leaf
tea_leaf!diffuse_
[Unknown]!0x0
[Broken backtraces]![Broken backtraces]

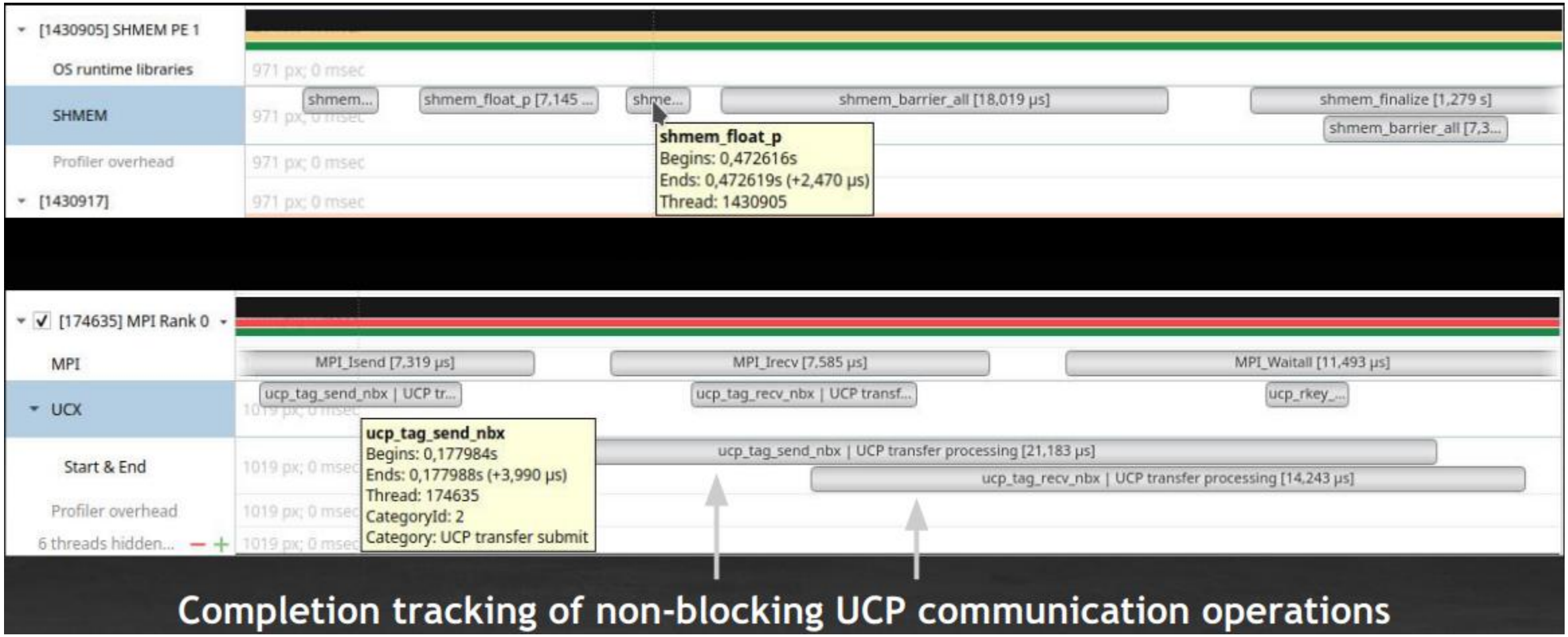
GPU METRIC SAMPLING



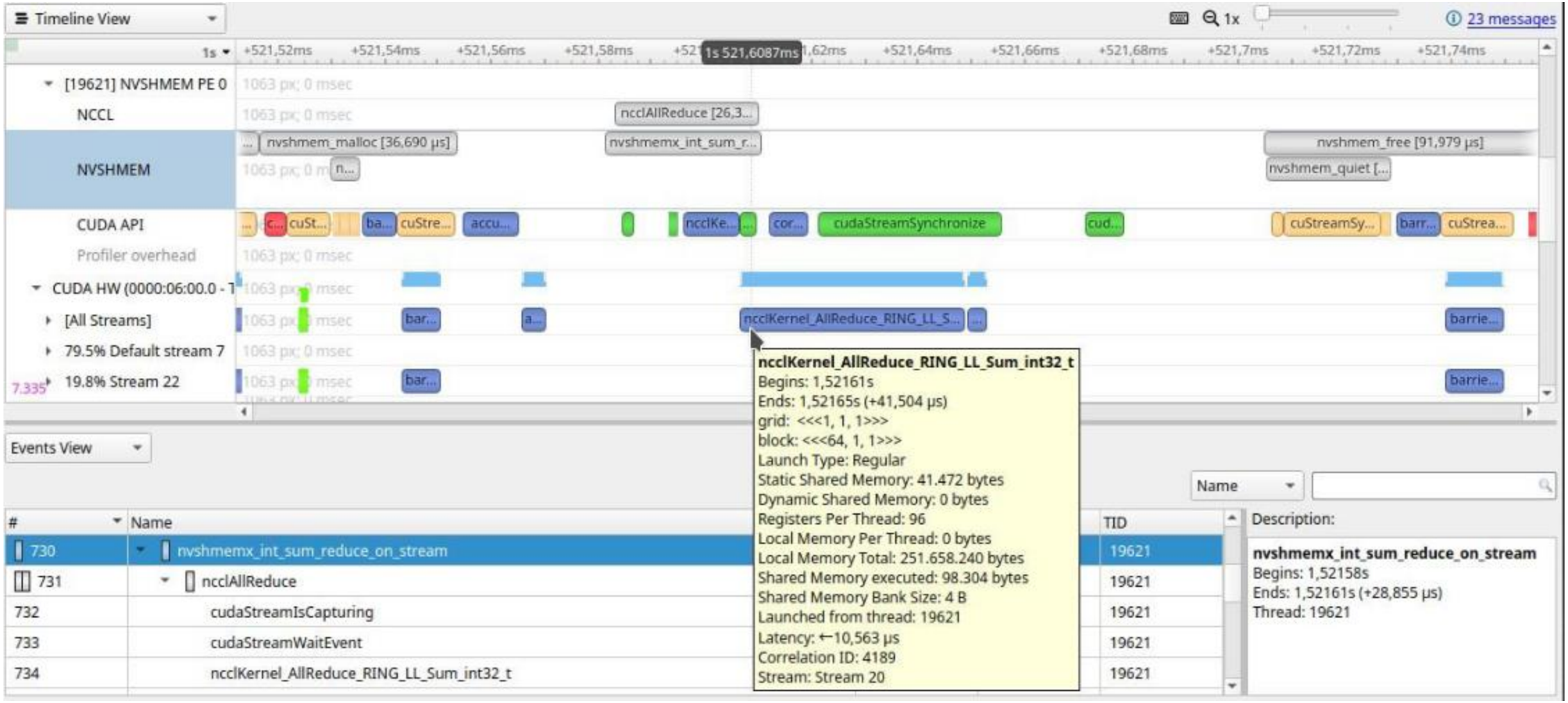
EXPERT SYSTEM



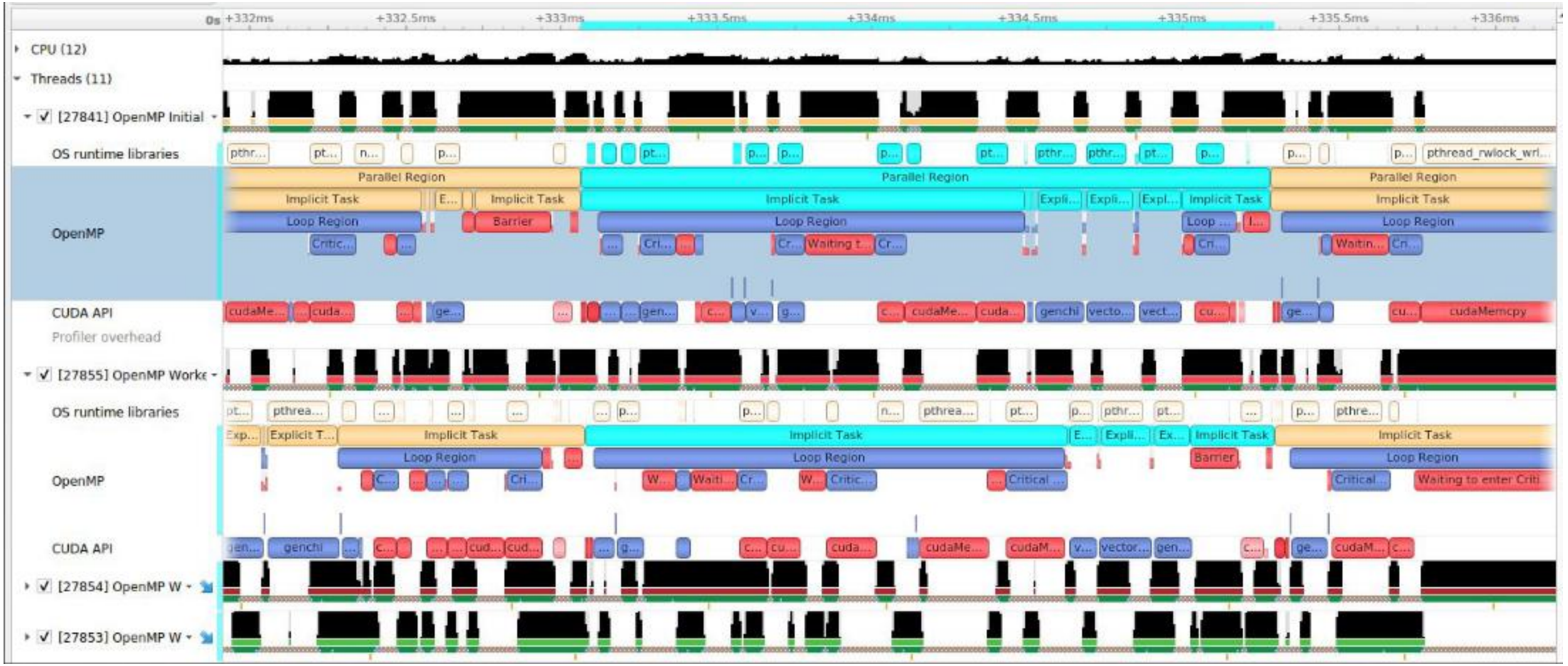
SHMEM, MPI AND UCX



NVSHMEM AND NCCL

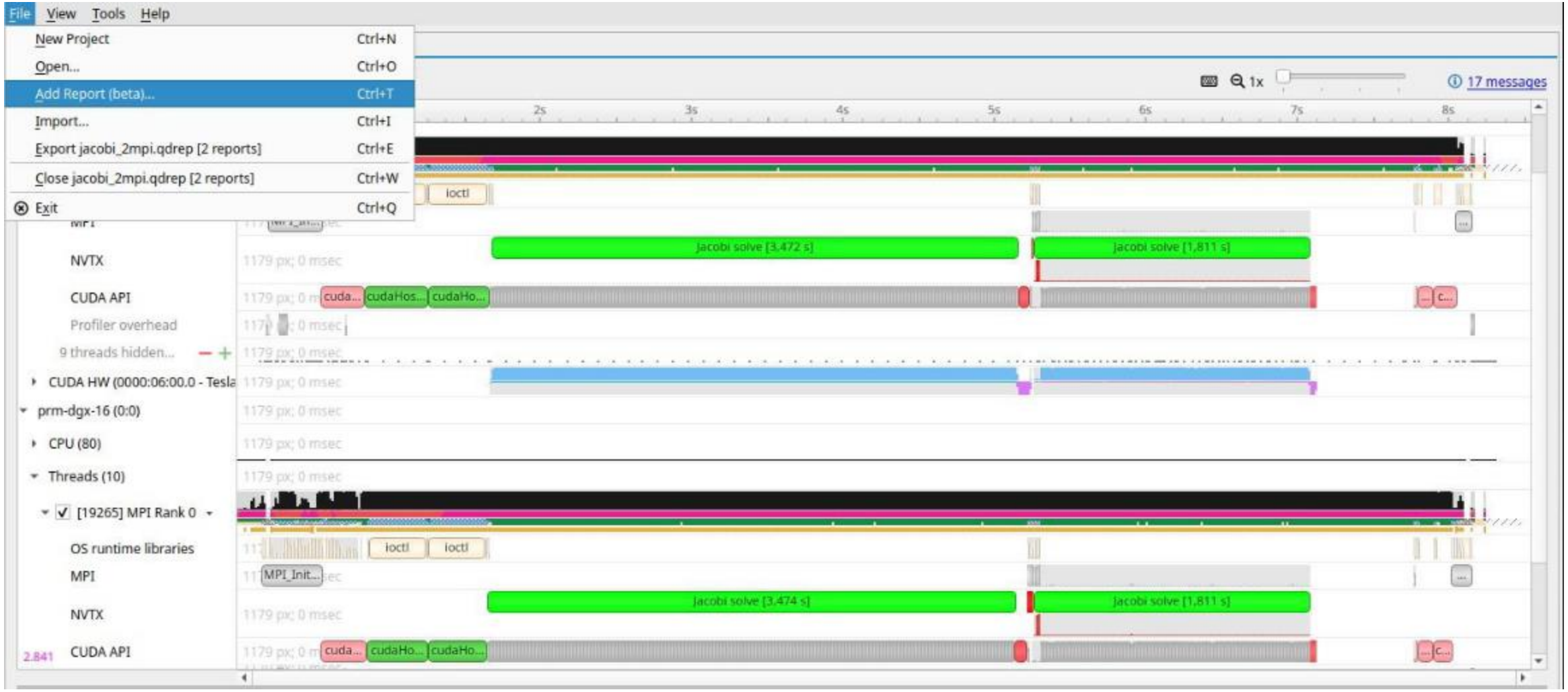


OPENMP



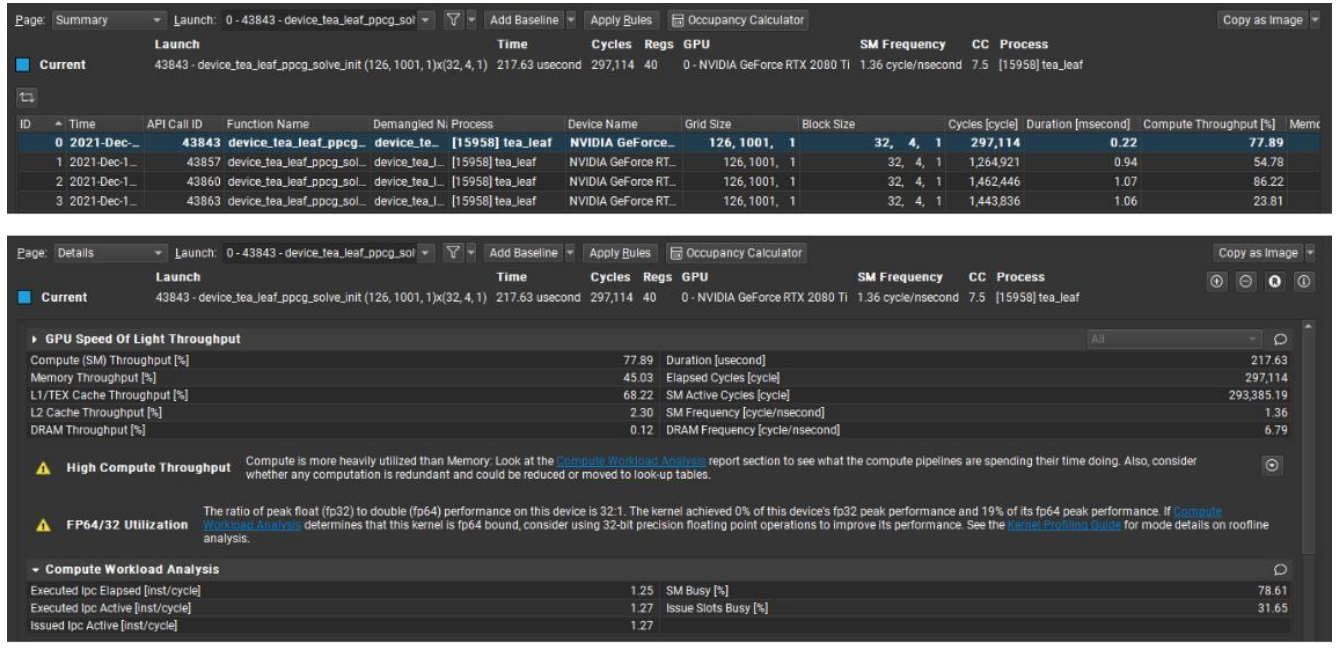
OMPT-capable OpenMP runtime required

LOADING MULTIPLE REPORTS



NSIGHT COMPUTE

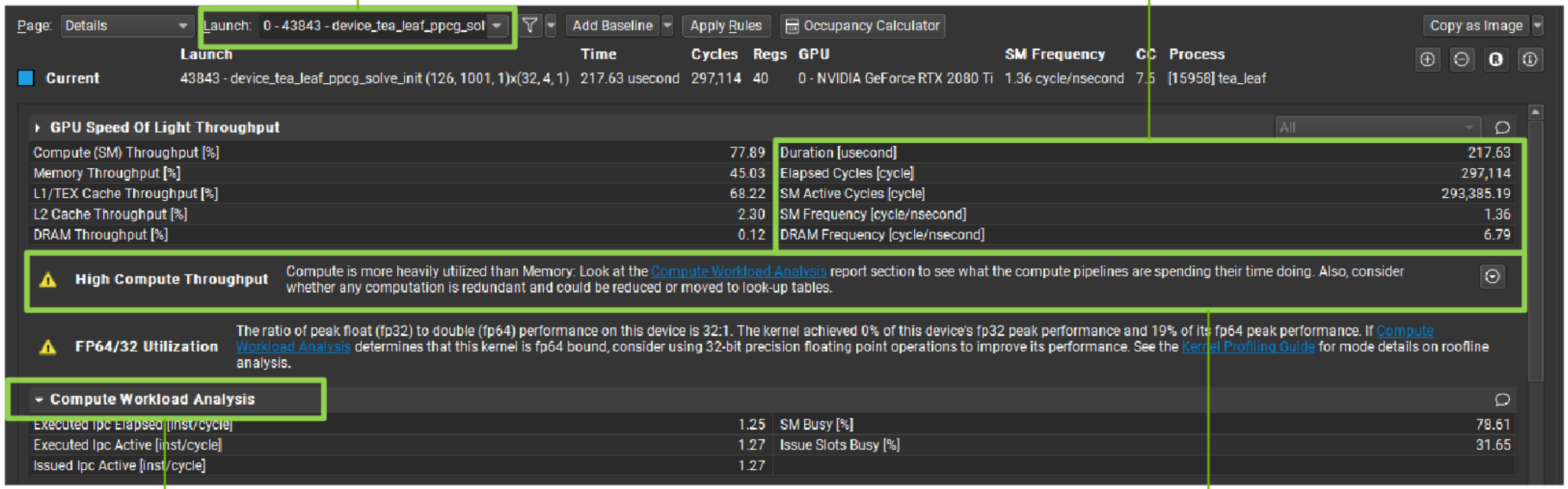
- Interactive CUDA kernel profiler
- Targeted metric sections for various performance aspects
- Customizable data collection and presentation (tables, charts, ...)
- GUI and CLI
- Python-based API for guided analysis and post-processing
- Support for remote profiling across machines and platforms



PROFILER REPORT

Selected result

Metric values

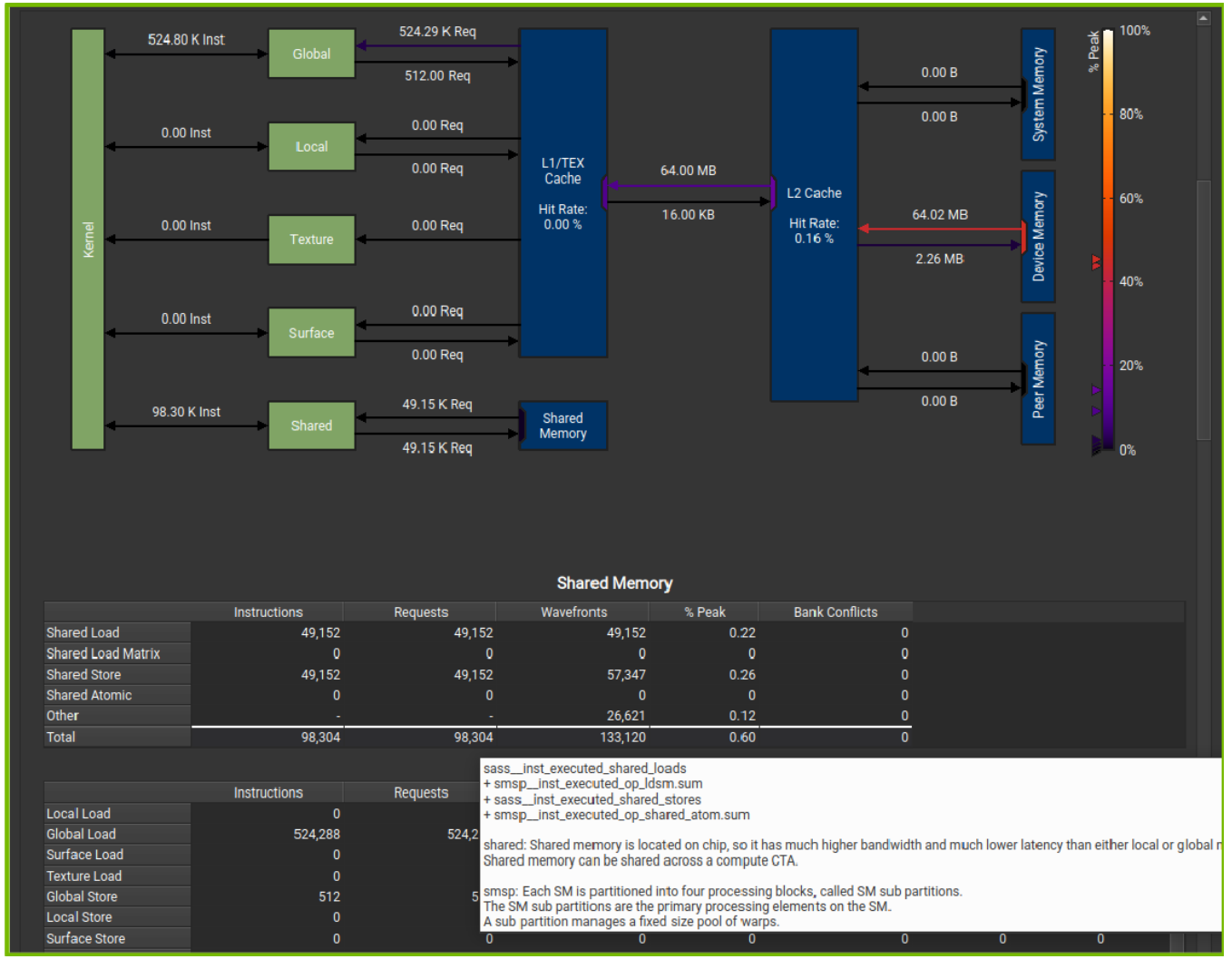


Expandable Sections

Expert Analysis (Rules)

DATA TRANSFER ANALYSIS

- Detailed memory workload analysis chart and tables
- Shows transferred data or throughputs
- Tooltips provide metric names, calculation formulas and detailed background info



Shared Memory

	Instructions	Requests	Wavefronts	% Peak	Bank Conflicts
Shared Load	49,152	49,152	49,152	0.22	0
Shared Load Matrix	0	0	0	0	0
Shared Store	49,152	49,152	57,347	0.26	0
Shared Atomic	0	0	0	0	0
Other	-	-	26,621	0.12	0
Total	98,304	98,304	133,120	0.60	0

	Instructions	Requests
Local Load	0	
Global Load	524,288	524,2
Surface Load	0	
Texture Load	0	
Global Store	512	5
Local Store	0	
Surface Store	0	

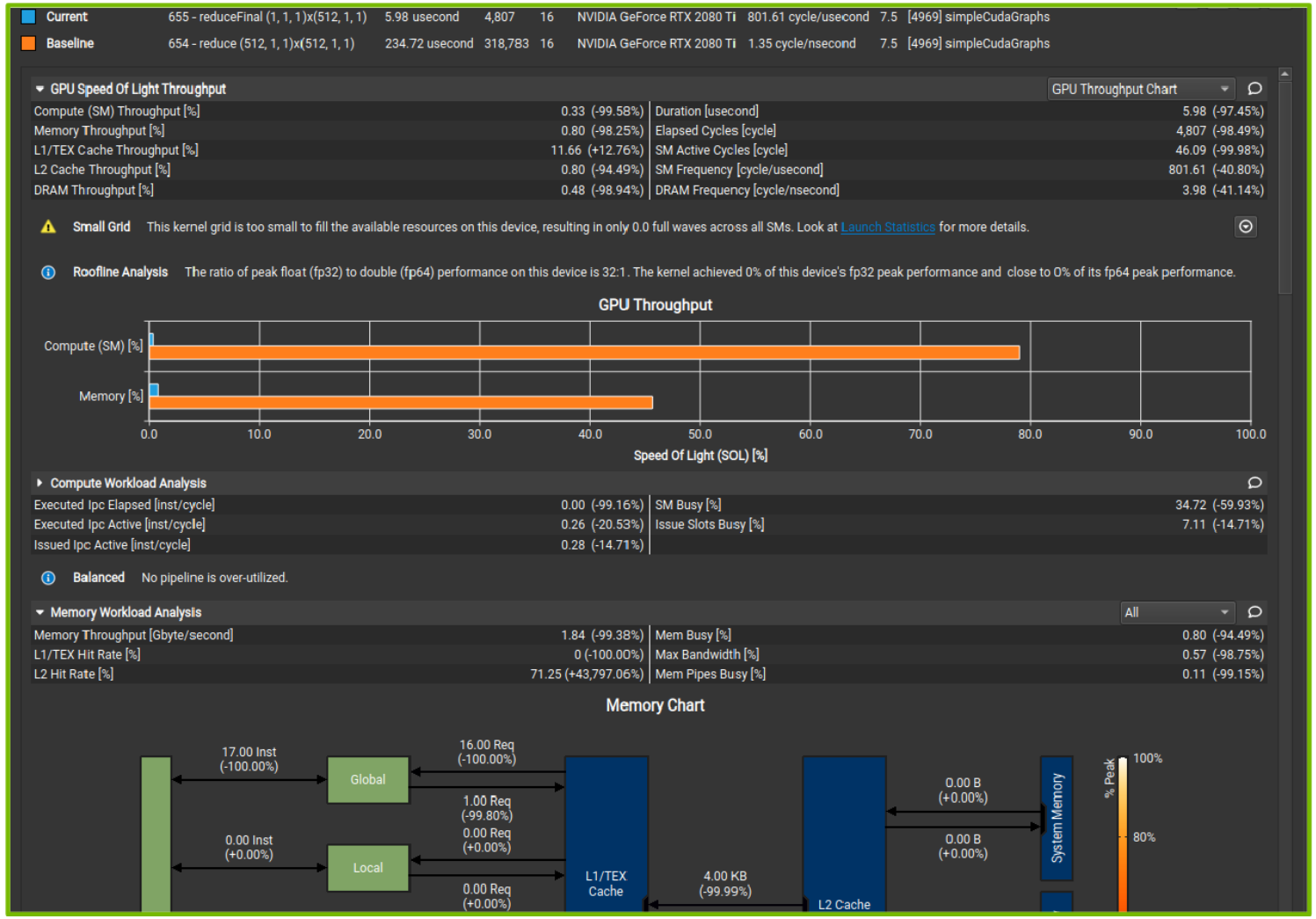
sass__inst_executed_shared_loads
+ smsp__inst_executed_op_ldsm.sum
+ sass__inst_executed_shared_stores
+ smsp__inst_executed_op_shared_atom.sum

shared: Shared memory is located on chip, so it has much higher bandwidth and much lower latency than either local or global m
Shared memory can be shared across a compute CTA.

smsp: Each SM is partitioned into four processing blocks, called SM sub partitions.
The SM sub partitions are the primary processing elements on the SM.
A sub partition manages a fixed size pool of warps.

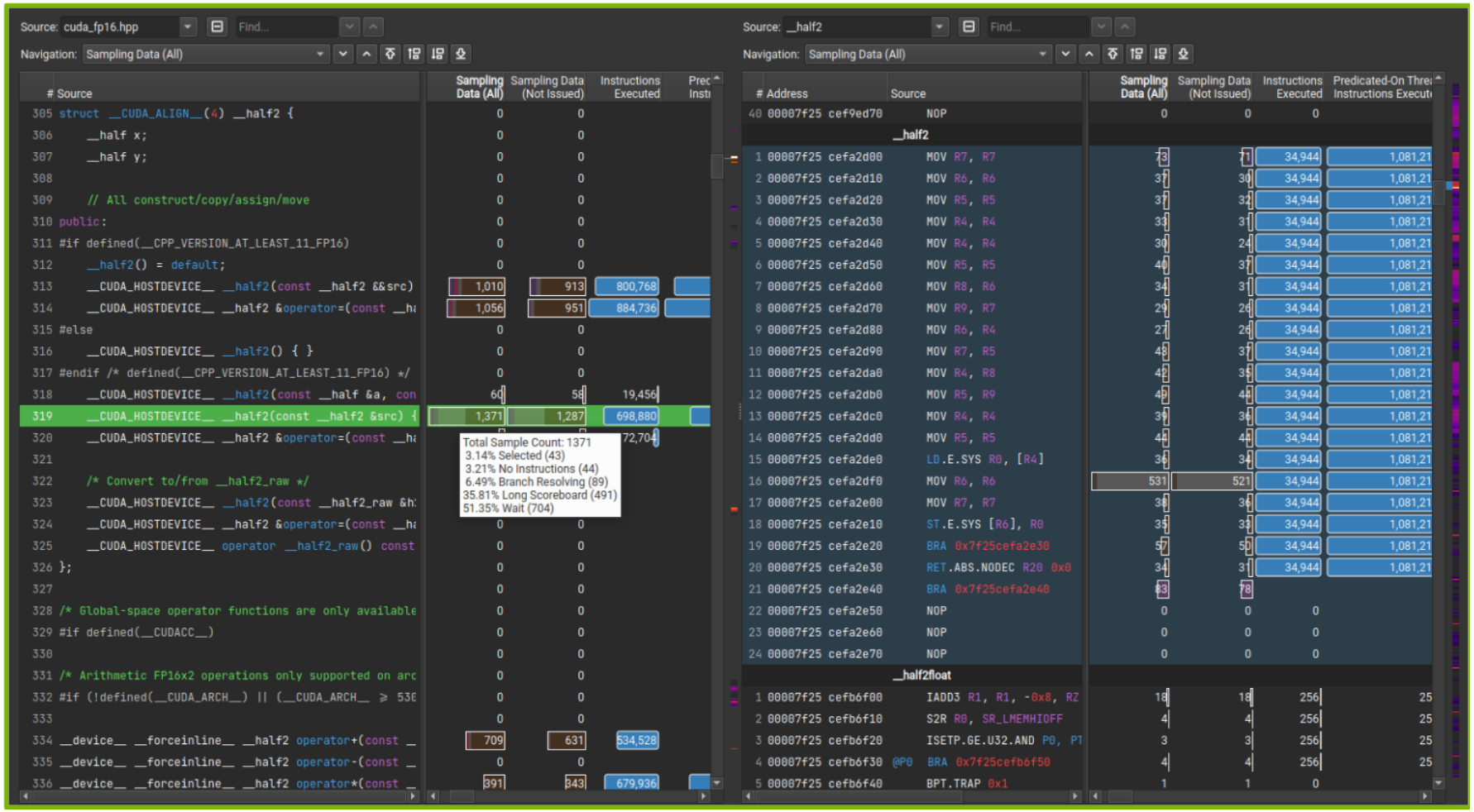
BASELINE COMPARISON

- Comparison of results directly within the tool with "Baselines“
- Supported across kernels, reports, and GPU architectures



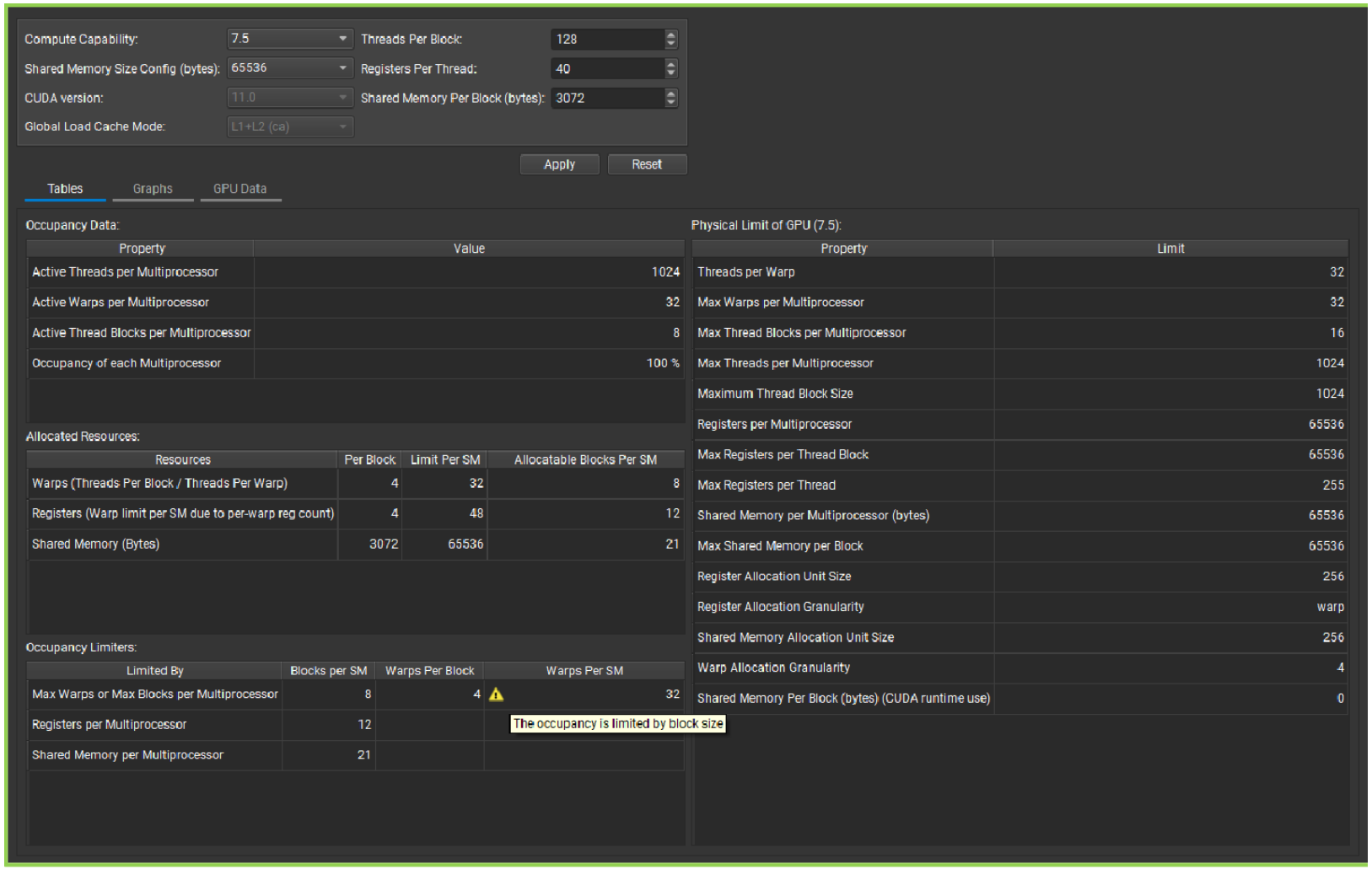
SOURCE CORRELATION

- Source/PTX/SASS analysis and correlation
- Source metrics per instruction and aggregated (e.g. PC sampling data)
- Metric heatmap



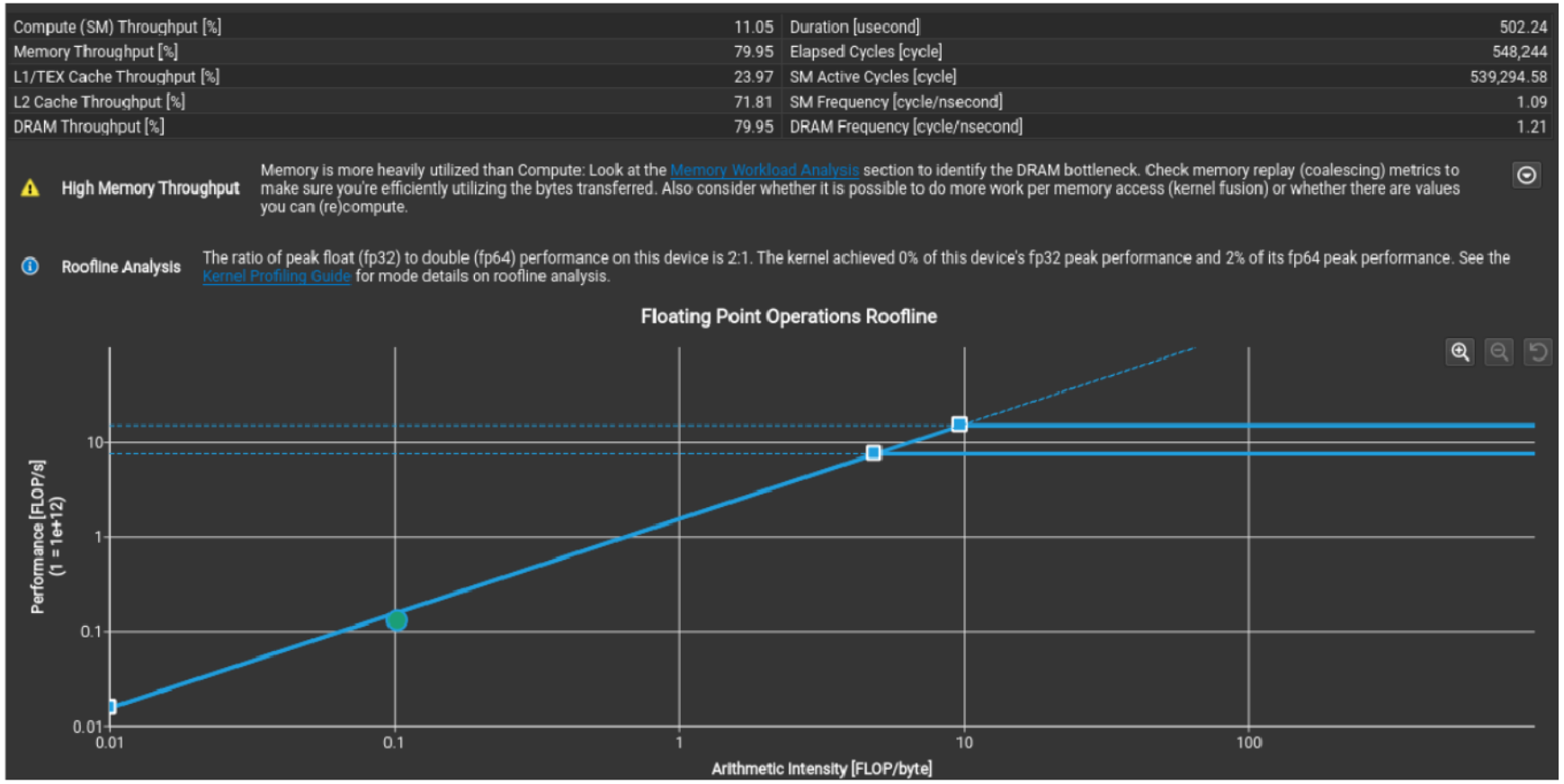
OCCUPANCY CALCULATOR

- In-depth analysis of GPU occupancy
- Quickly determine if computational rescources are wasted



ROOFLINE ANALYSIS

- Determine whether the application is memory bound or compute bound
- Guided analysis points to detailed analysis of the most severe problem



QUESTIONS?

JÜLICH Forschungszentrum | JÜLICH SUPERCOMPUTING CENTRE

scalasca

- http://www.scalasca.org
- scalasca@fz-juelich.de

Score-P

Scalable performance measurement infrastructure for parallel codes

- http://www.score-p.org
- support@score-p.org