



GPU PERFORMANCE ANALYSIS SCORE-P AND NVIDIA TOOLS

APRIL 8, 2022 | MICHAEL KNOBLOCH

GPU Analysis using the **SCORE-P ECOSYSTEM**

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`)

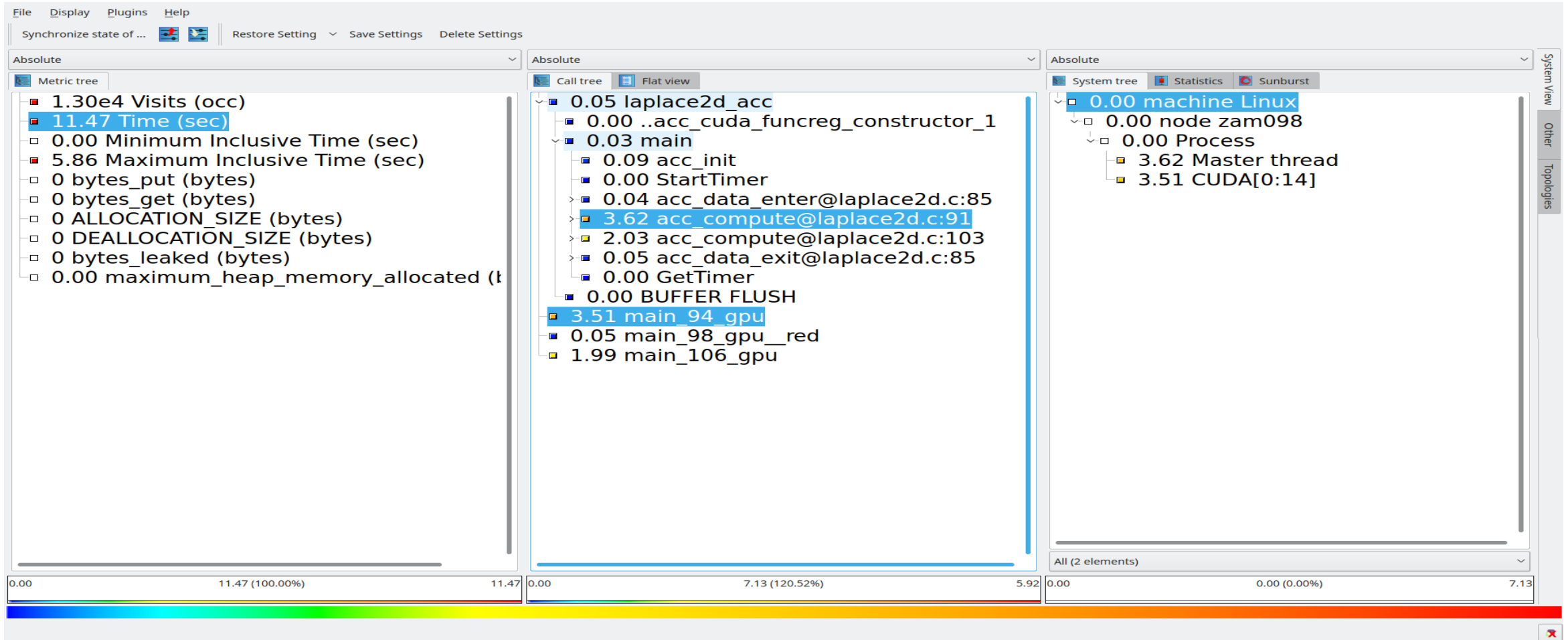
EXAMPLE: OPENACC

The screenshot displays a performance analysis tool interface with three main panels. The left panel, titled 'Metric tree', shows a list of metrics: 1.00e4 Visits (occ), 6.61 Time (sec) (highlighted), 0.00 Minimum Inclusive Time (sec), 6.61 Maximum Inclusive Time (sec), 0 bytes_put (bytes), 0 bytes_get (bytes), 0 ALLOCATION_SIZE (bytes), 0 DEALLOCATION_SIZE (bytes), 0 bytes_leaked (bytes), and 0.00 maximum_heap_memory_allocate. The middle panel, titled 'Call tree', shows a hierarchical view of the program's execution: 0.04 laplace2d_acc, 0.00 ..acc_cuda_funcreg_constructor_1, 0.05 main, 0.05 acc_init, 0.00 StartTimer, 0.05 acc_data_enter@laplace2d.c:85, 4.16 acc_compute@laplace2d.c:91 (highlighted), 2.21 acc_compute@laplace2d.c:103, 0.05 acc_data_exit@laplace2d.c:85, and 0.00 GetTimer. A blue callout box with the text 'Where are my kernels?' points to the 'acc_compute' entries. The right panel shows the source code of the program, with lines 79-109 visible. The code includes OpenACC directives: `#pragma acc data copy(A, Anew)`, `#pragma omp parallel for shared(m, n, Anew, A)`, and `#pragma acc kernels`. The code also includes a loop for calculating the Laplace 2D function.

Where are my kernels?

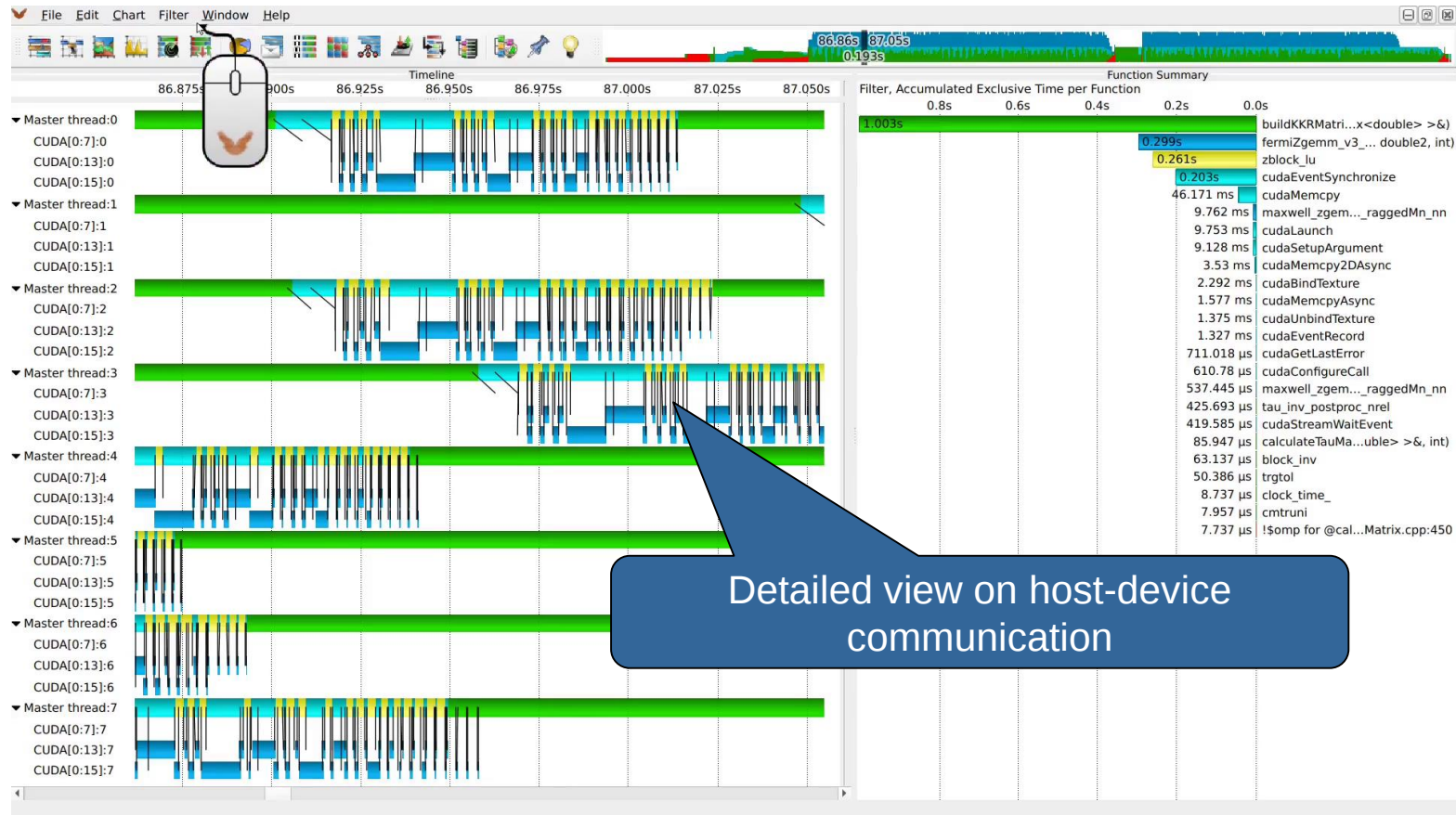
Pure OpenACC measurements contain host-side events only

EXAMPLE: OPENACC + CUDA



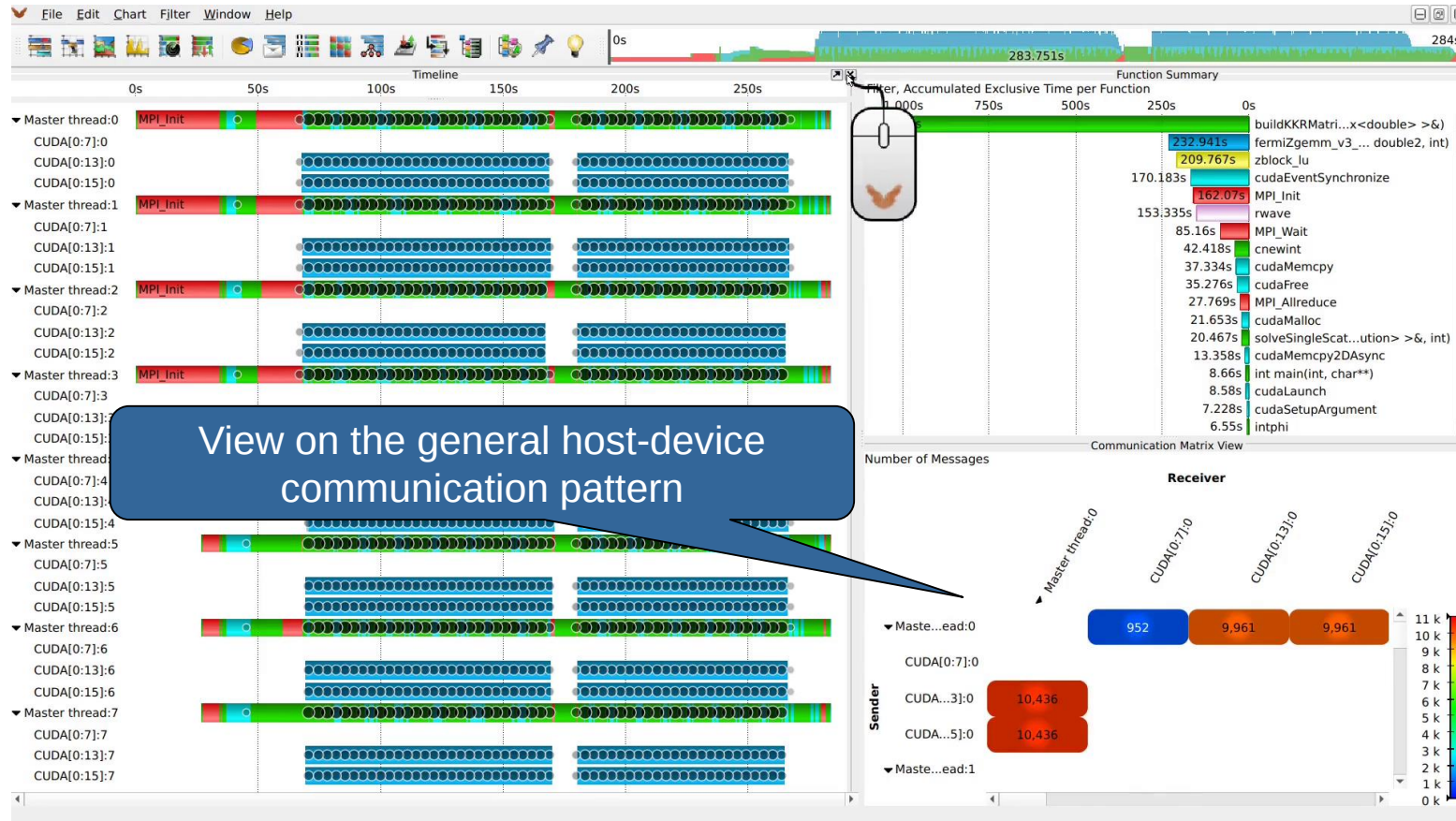
Enabling CUDA also shows kernels on the GPU

VAMPIR CUDA EXAMPLE



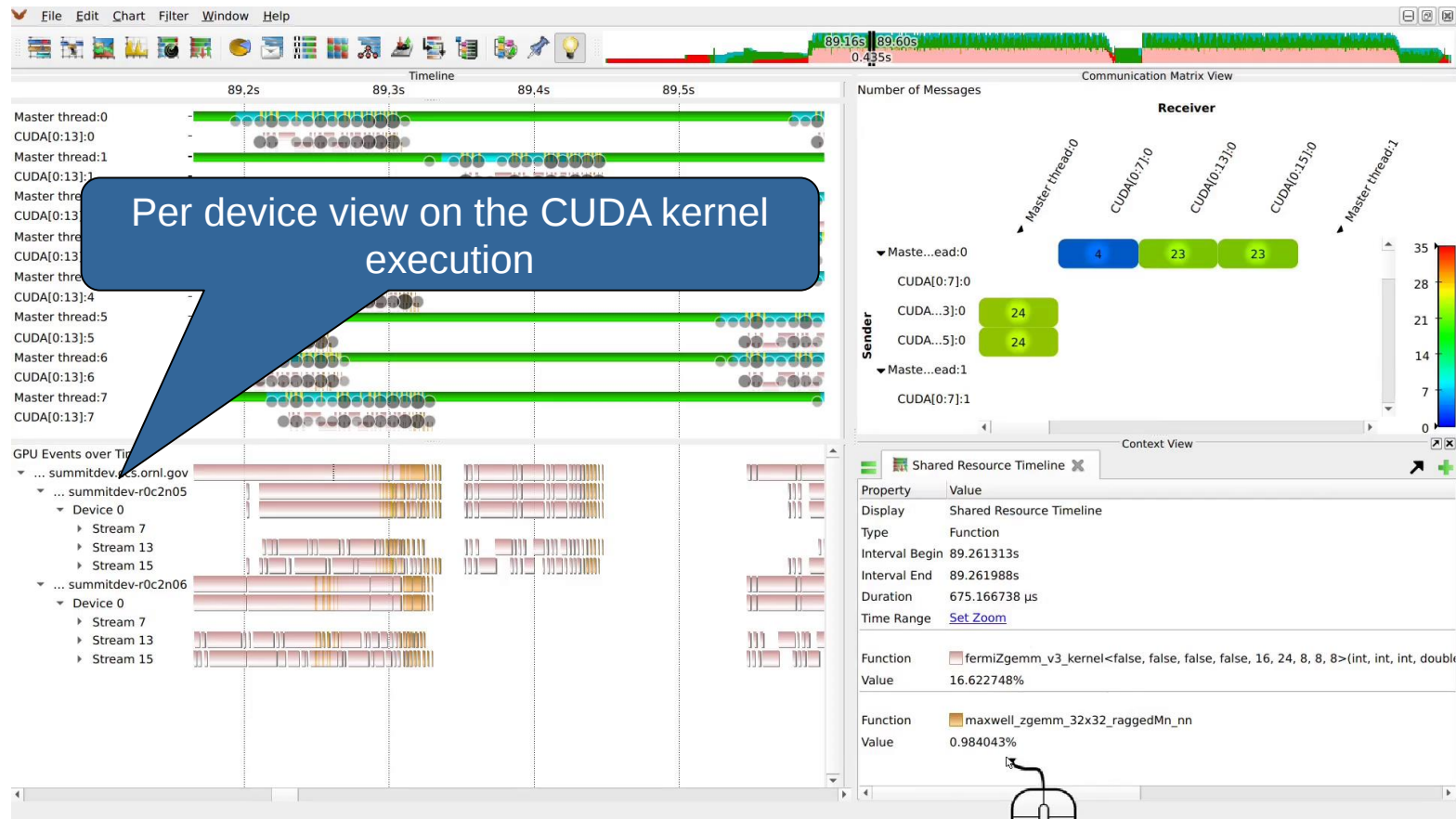
- Material science code LSMS
- CUDA is utilized for heavy computations
- CUDA streams are child's of the owning Process
- Allows an in-depth analysis of host-device communication

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?

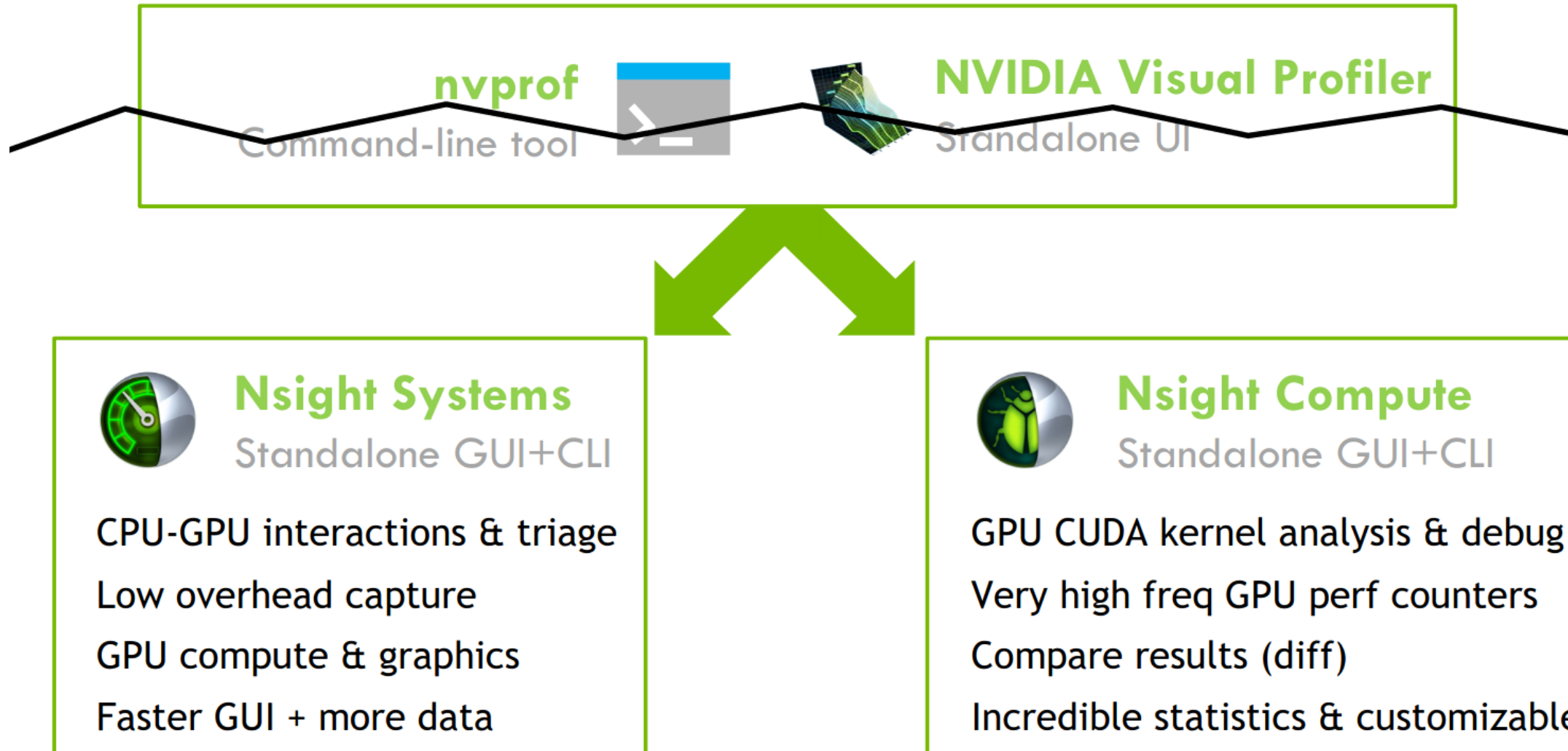
- The Scalasca trace analyzer aborts if it encounters non-CPU locations, e.g. GPU streams
- However, GPU events are not too important for the analysis performed by Scalasca
- To enable a Scalasca analysis of a GPU application, either:
 - set SCOREP_CUDA_ENABLE=0 to disable all GPU events
 - set SCOREP_CUDA_ENABLE=runtime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

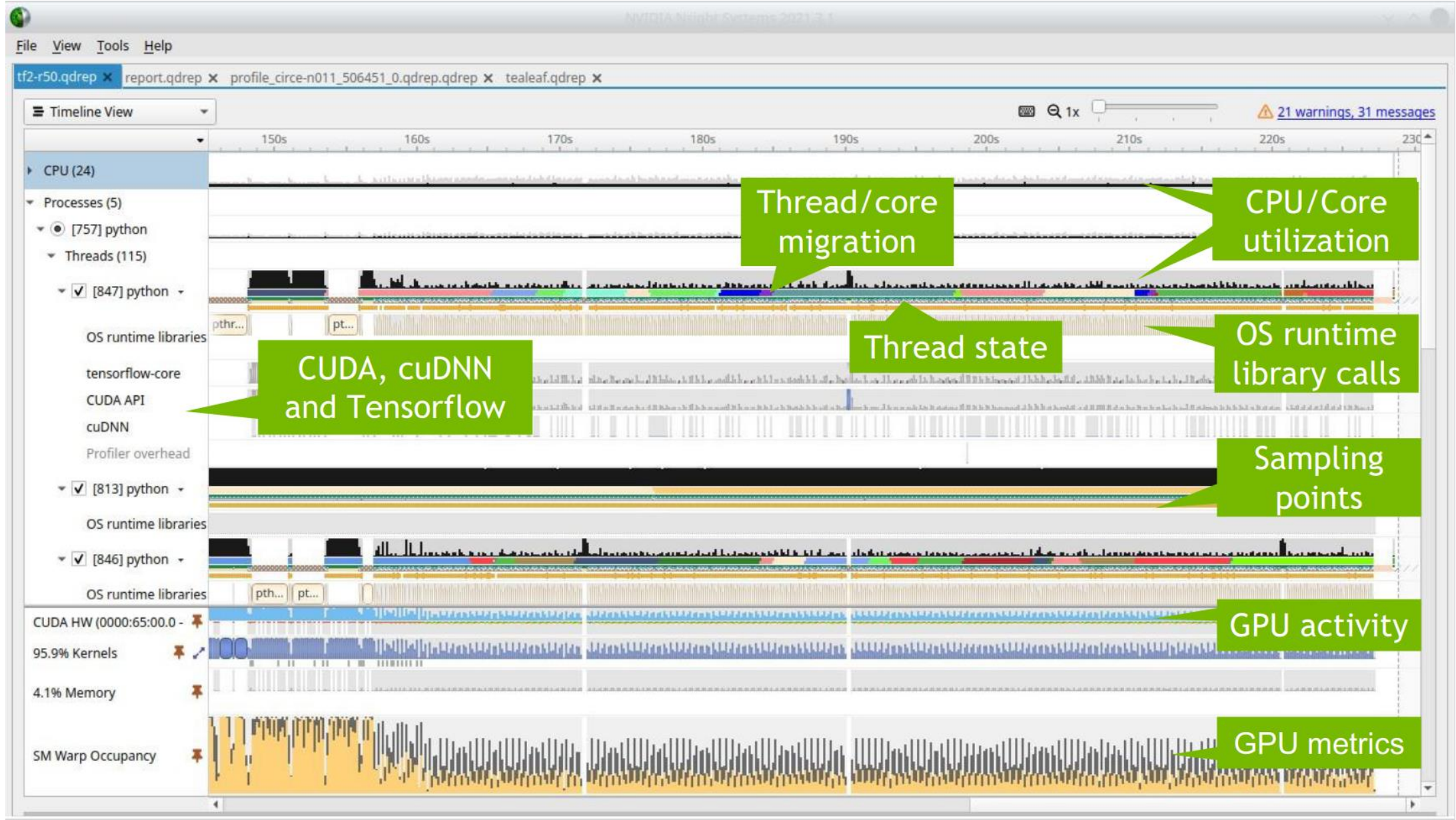
GPU Analysis using **NVIDIA NSIGHT**

LEGACY TRANSITION



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)



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.

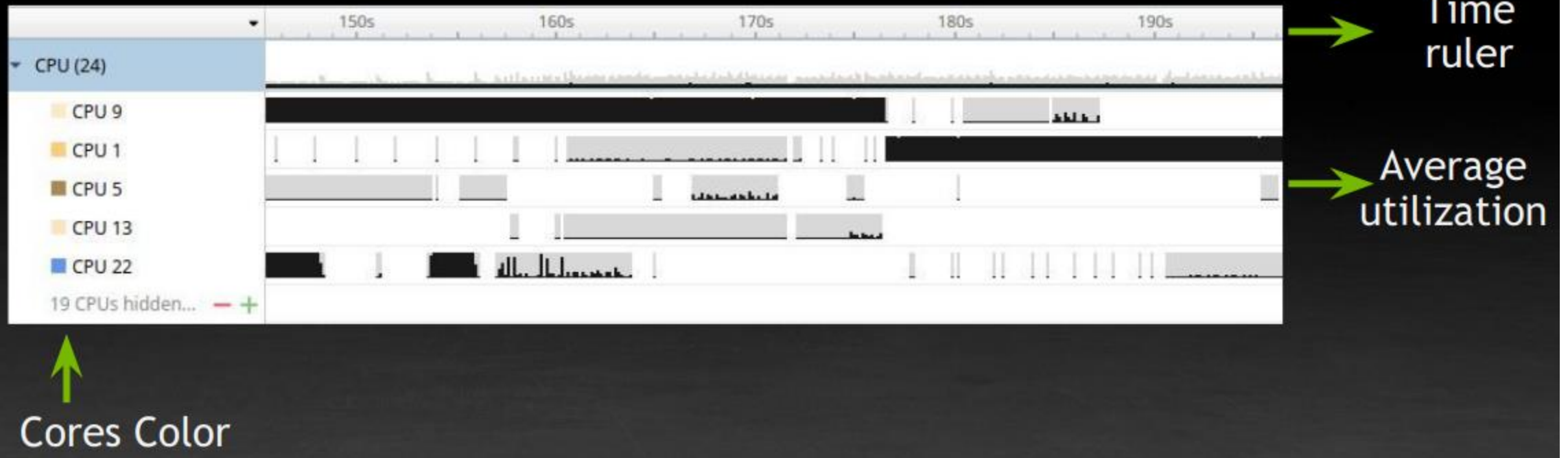
Profile only specific ranks. ⇒

```
#!/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
```

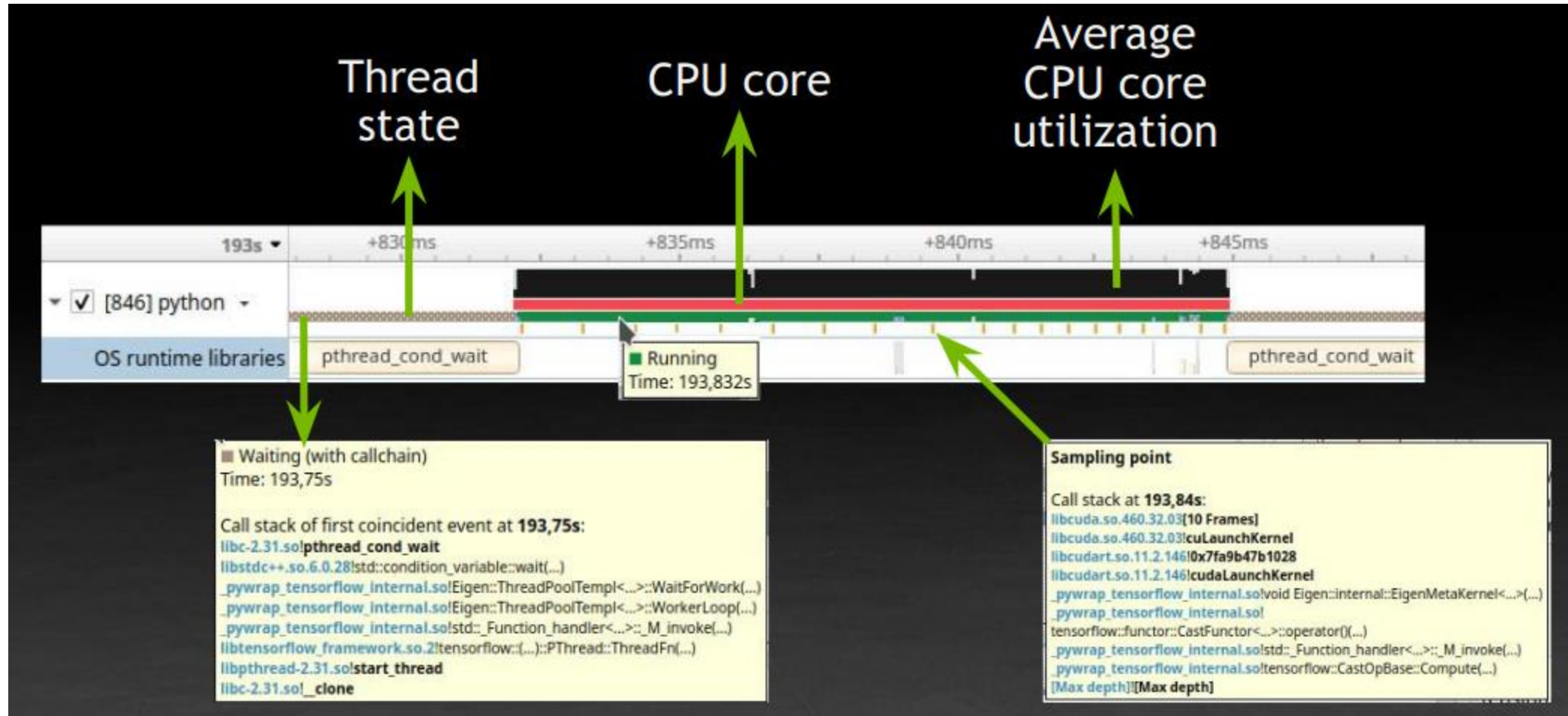
CPU CORES WORKLOAD

See CPU core utilization by application's threads

Locate idle time on CPU cores



CPU THREAD ACTIVITY

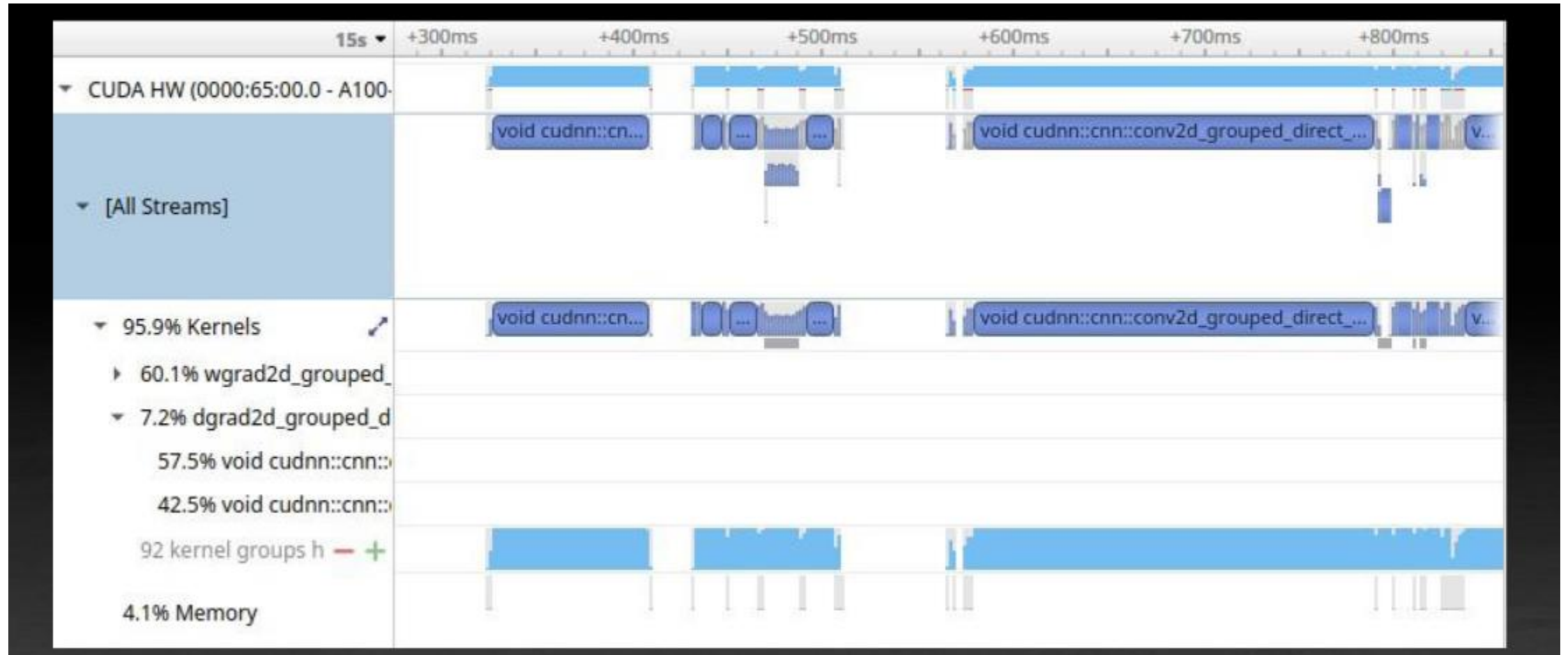


CUDA API

- 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



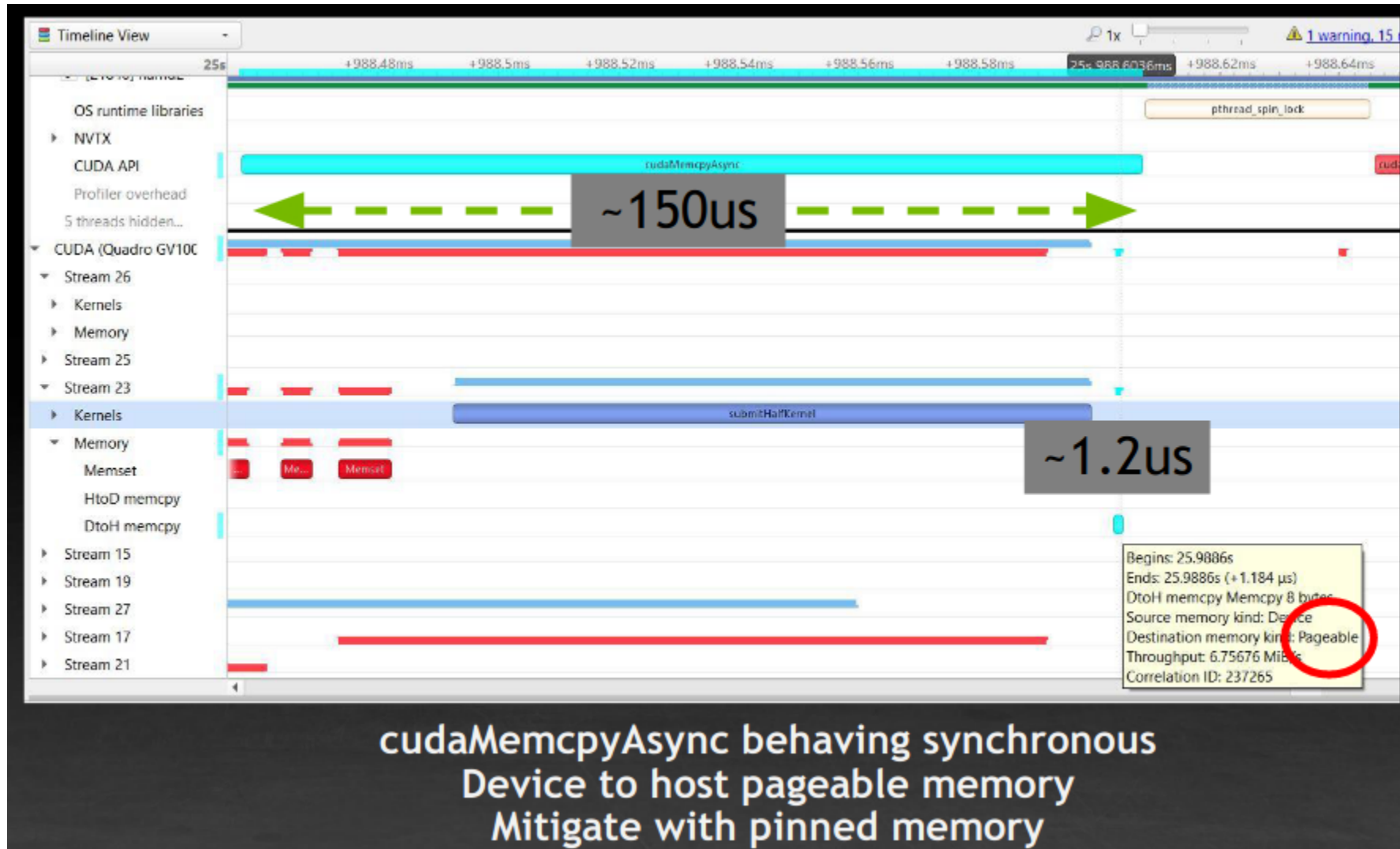
GPU UTILIZATION



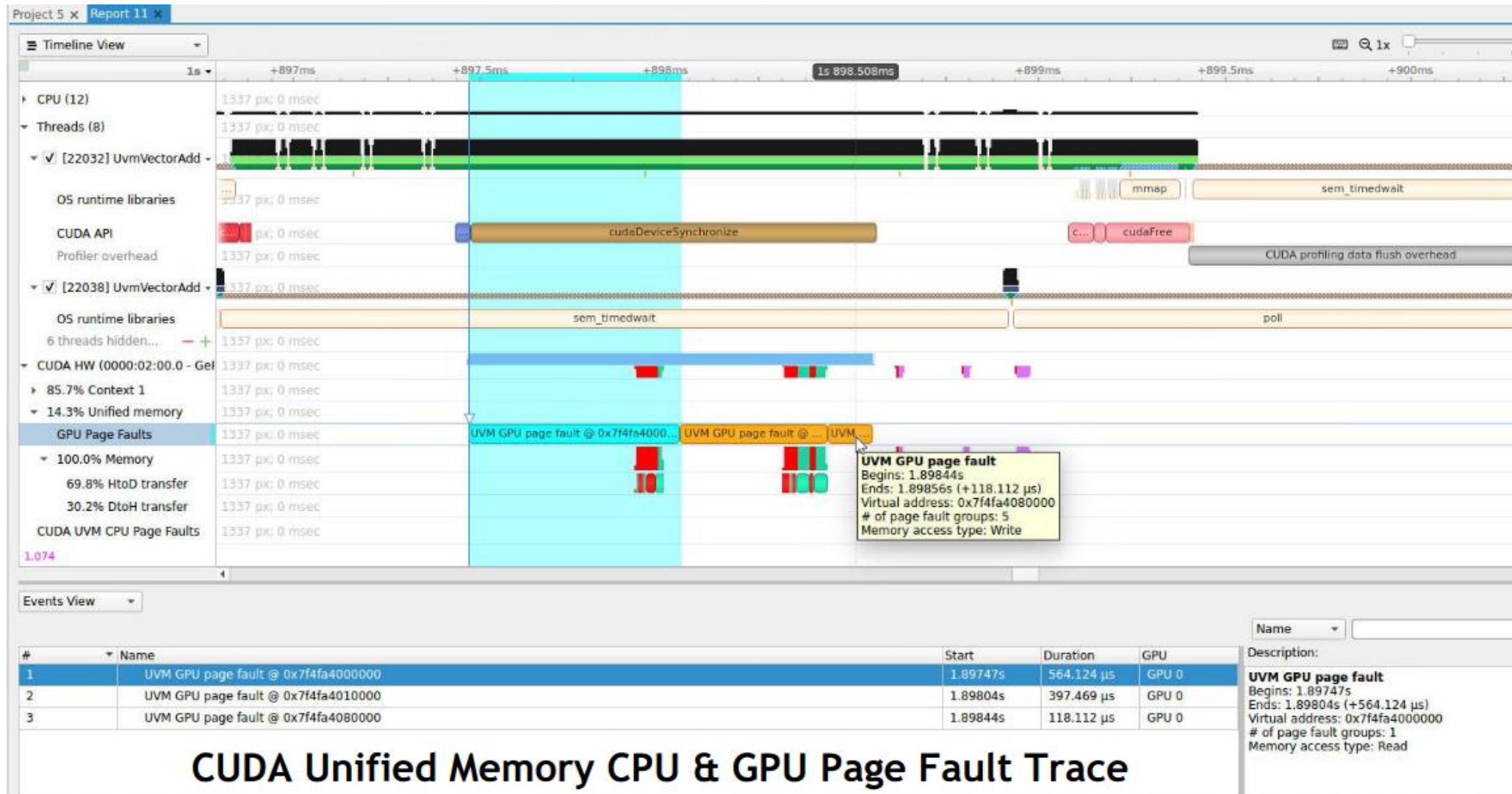
GPU UTILIZATION IN DETAIL



CUDA MEMORY TRANSFERS

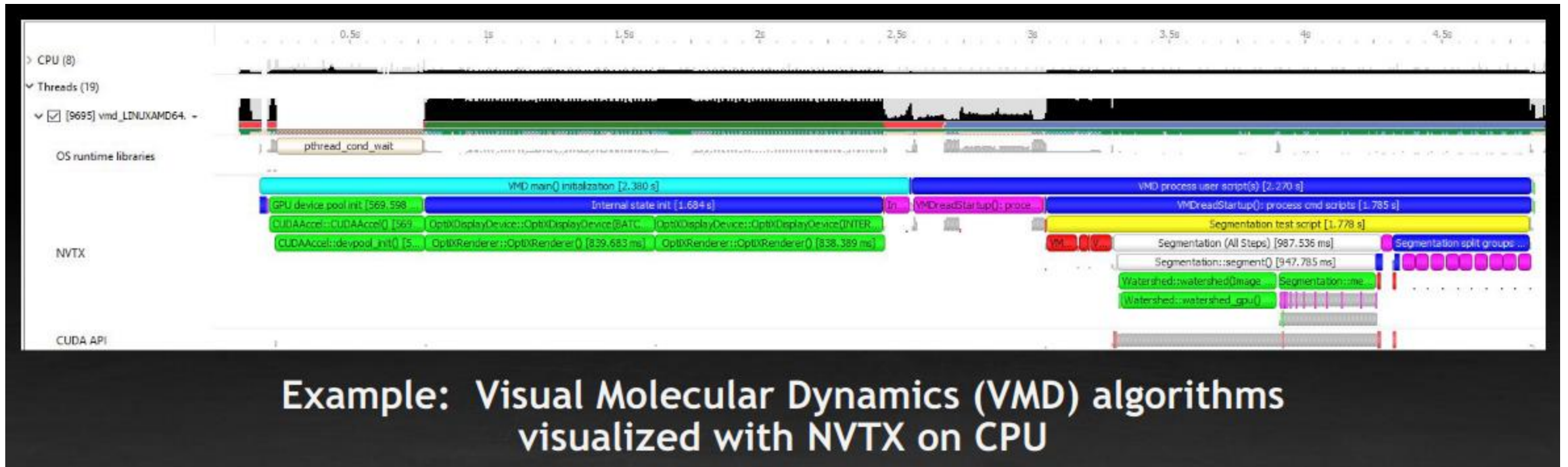


UNIFIED MEMORY ANALYSIS

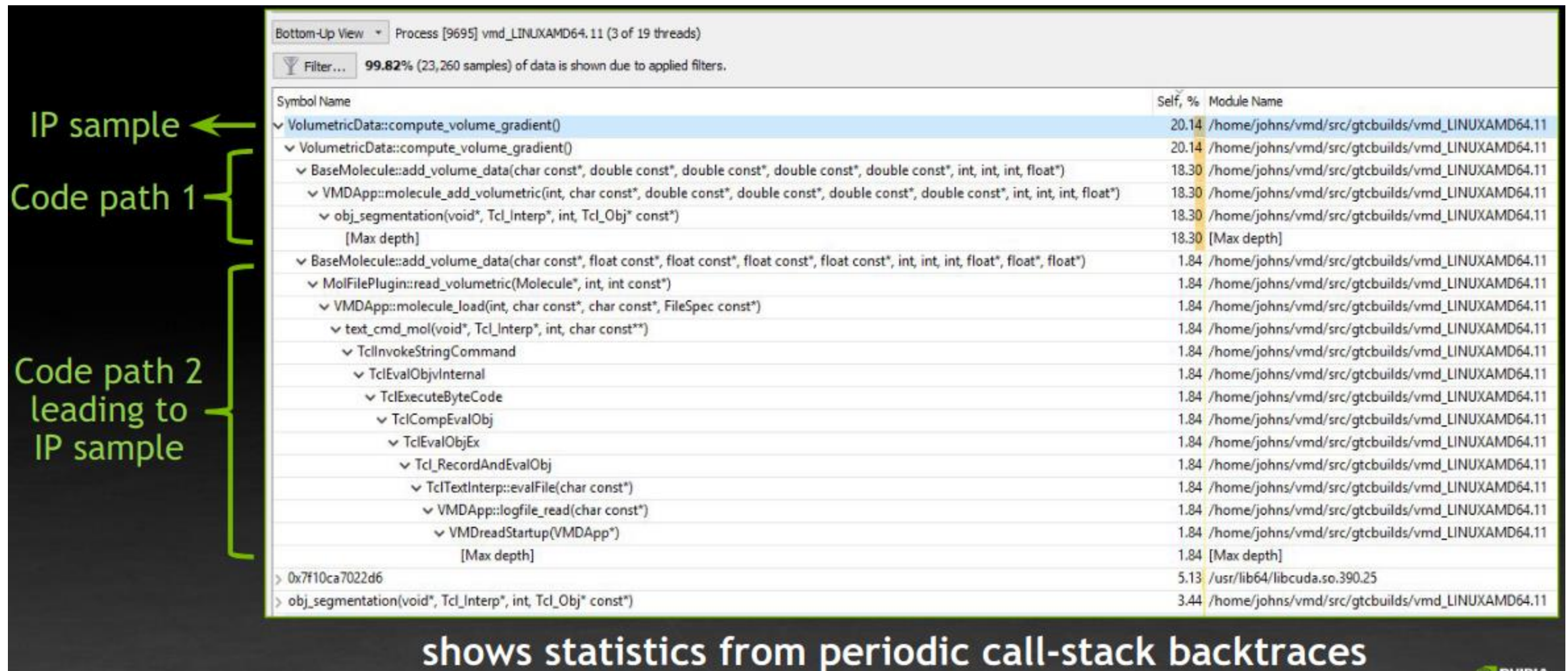


CUDA Unified Memory CPU & GPU Page Fault Trace

USER ANNOTATIONS WITH NVTX



STATISTICAL SAMPLING



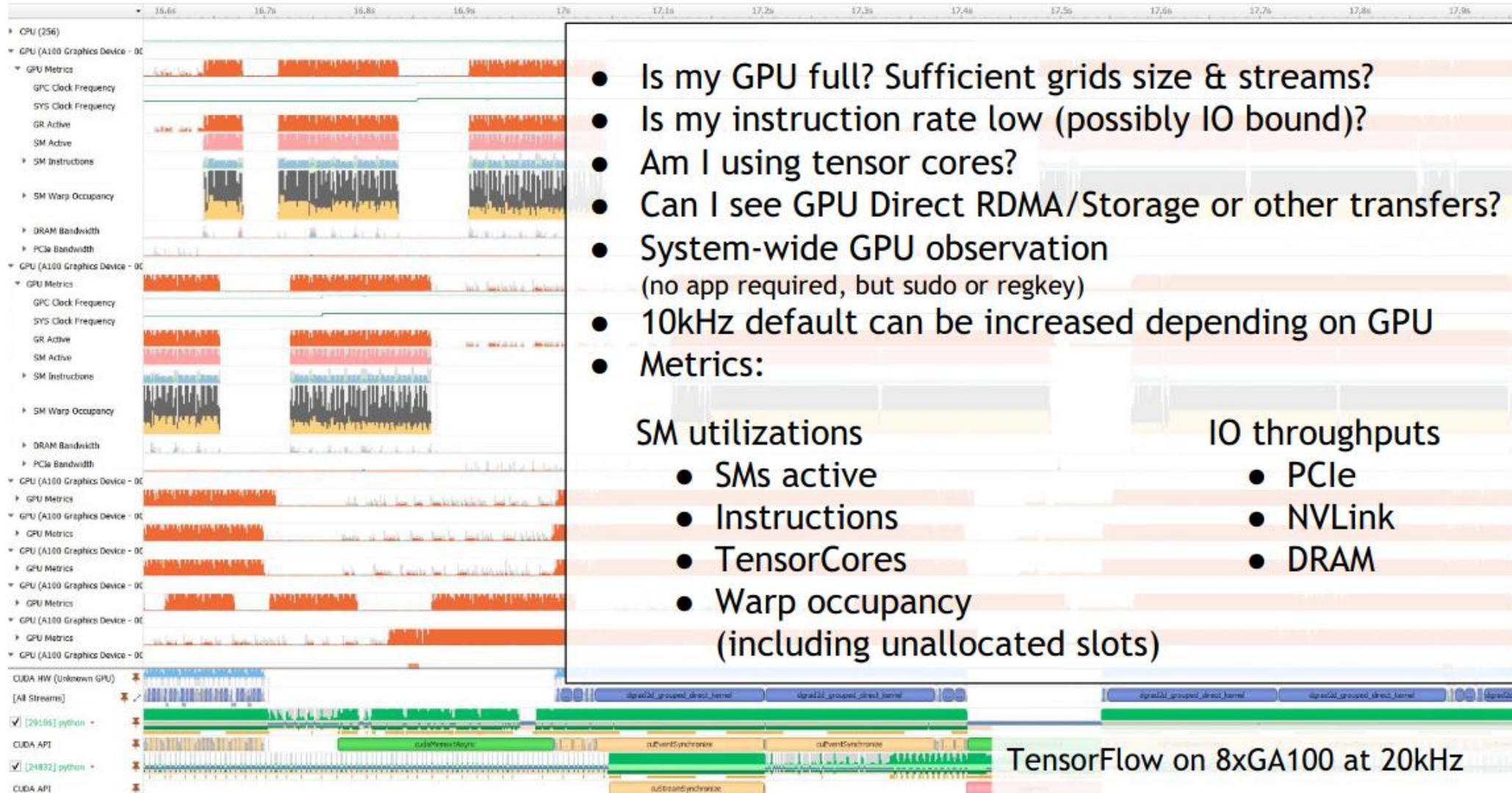
EVENTS VIEW

Events View

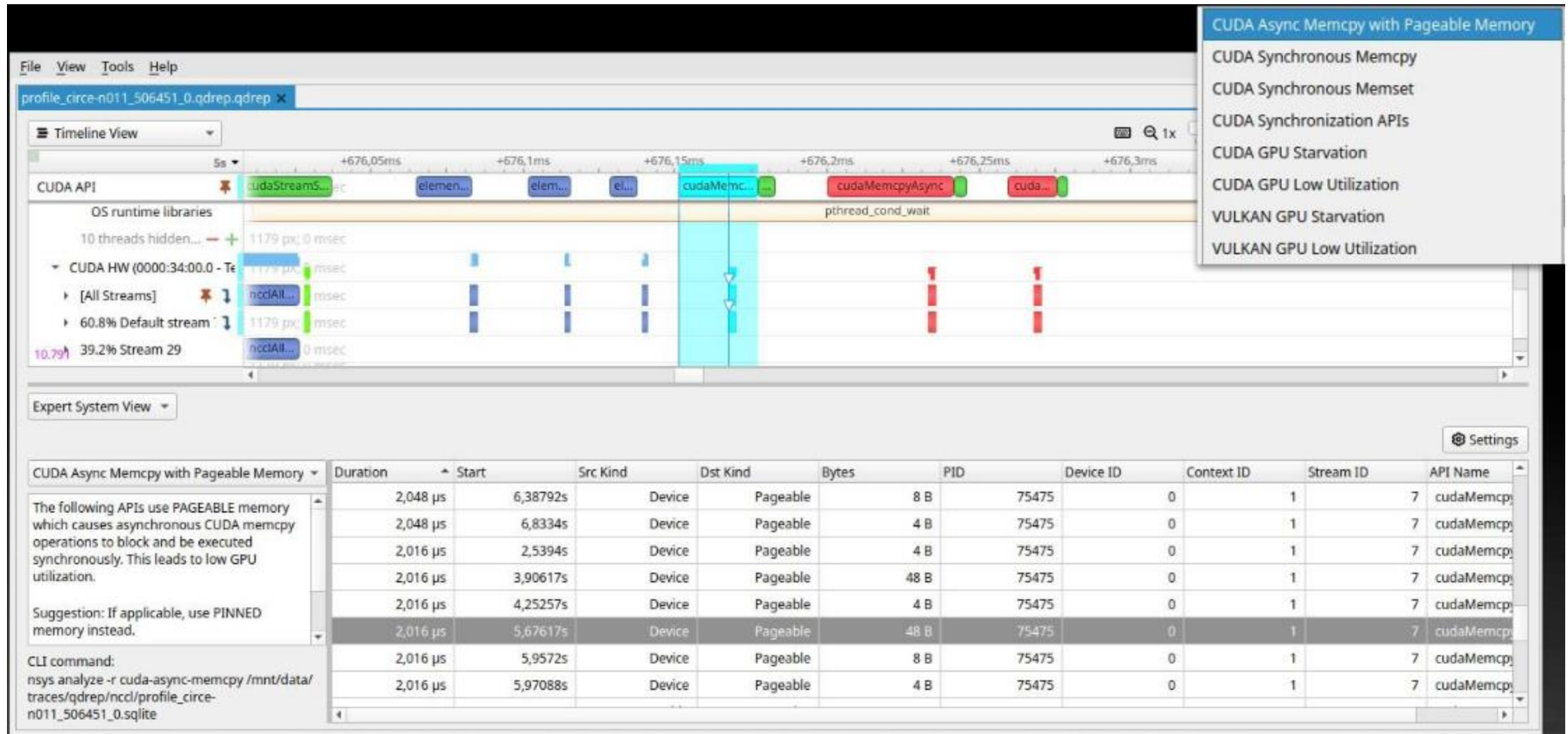
Name

#	Name	Start	Duration	TID	Description:
149153	device_tea_leaf_ppcg_solve_calc_sd_new	11,4088s	3,265 μs	390019	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]
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	

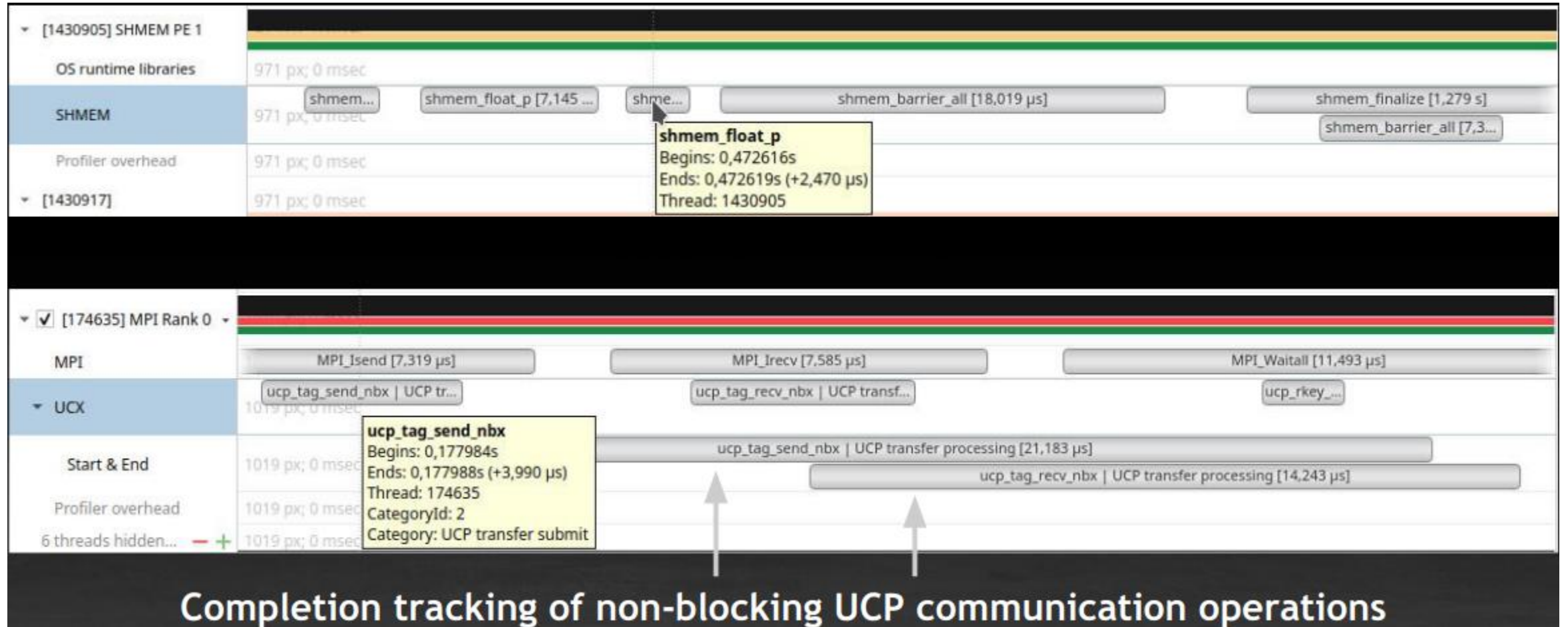
GPU METRIC SAMPLING



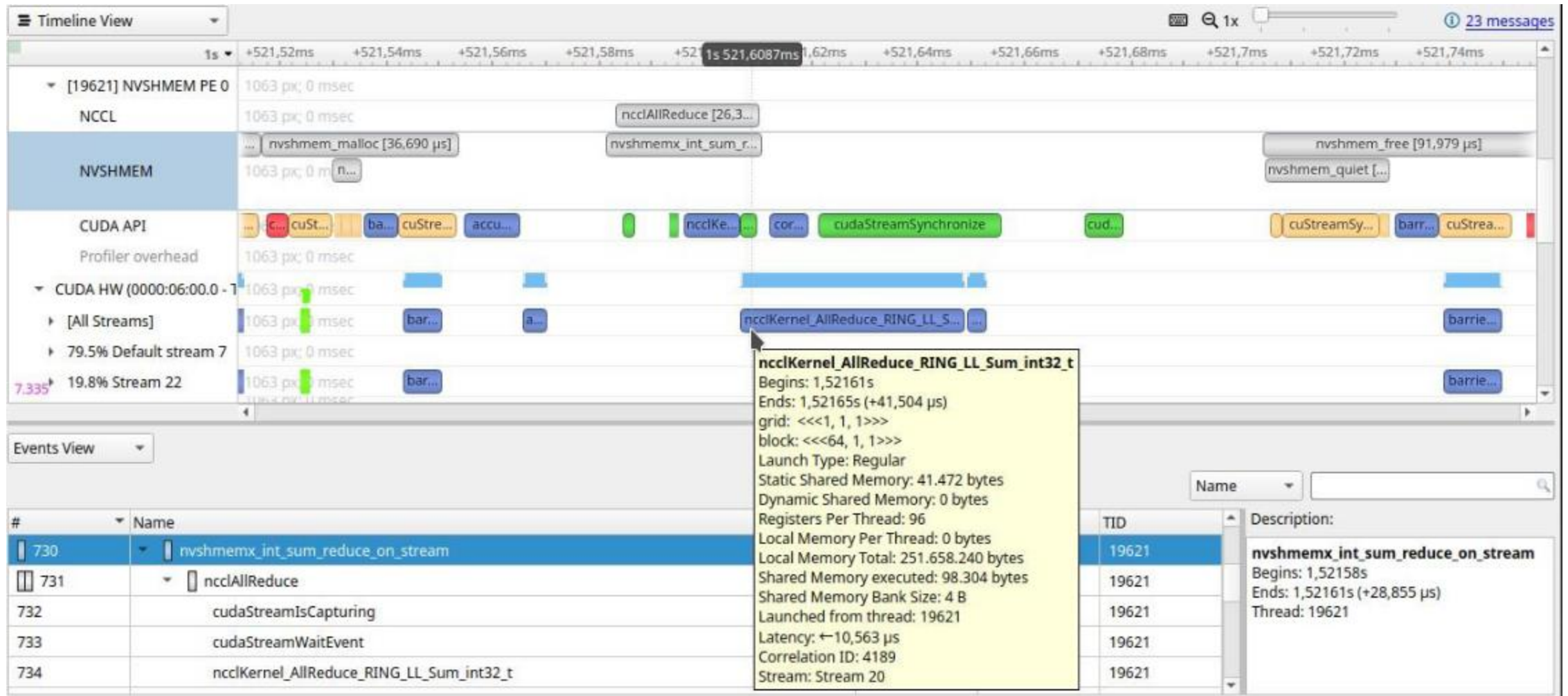
EXPERT SYSTEM



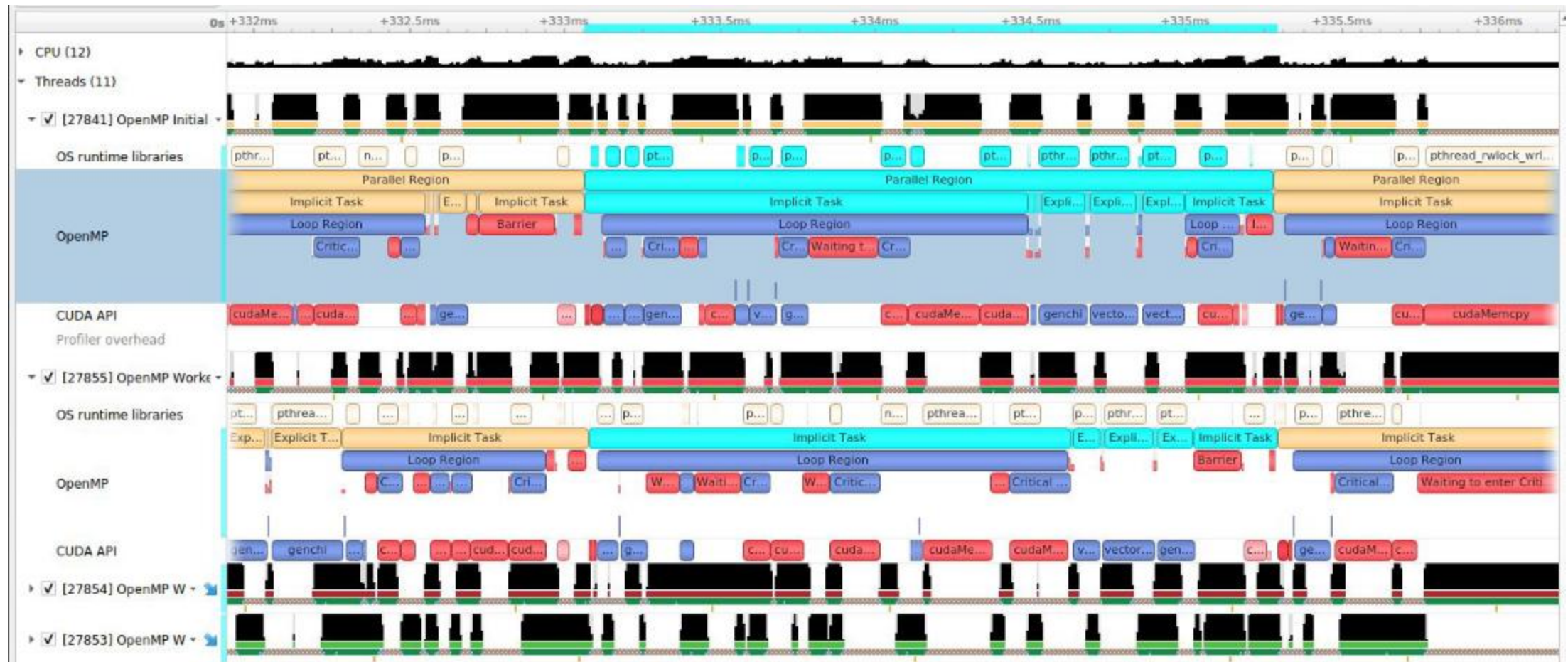
SHMEM, MPI AND UCX



NVSHMEM AND NCCL

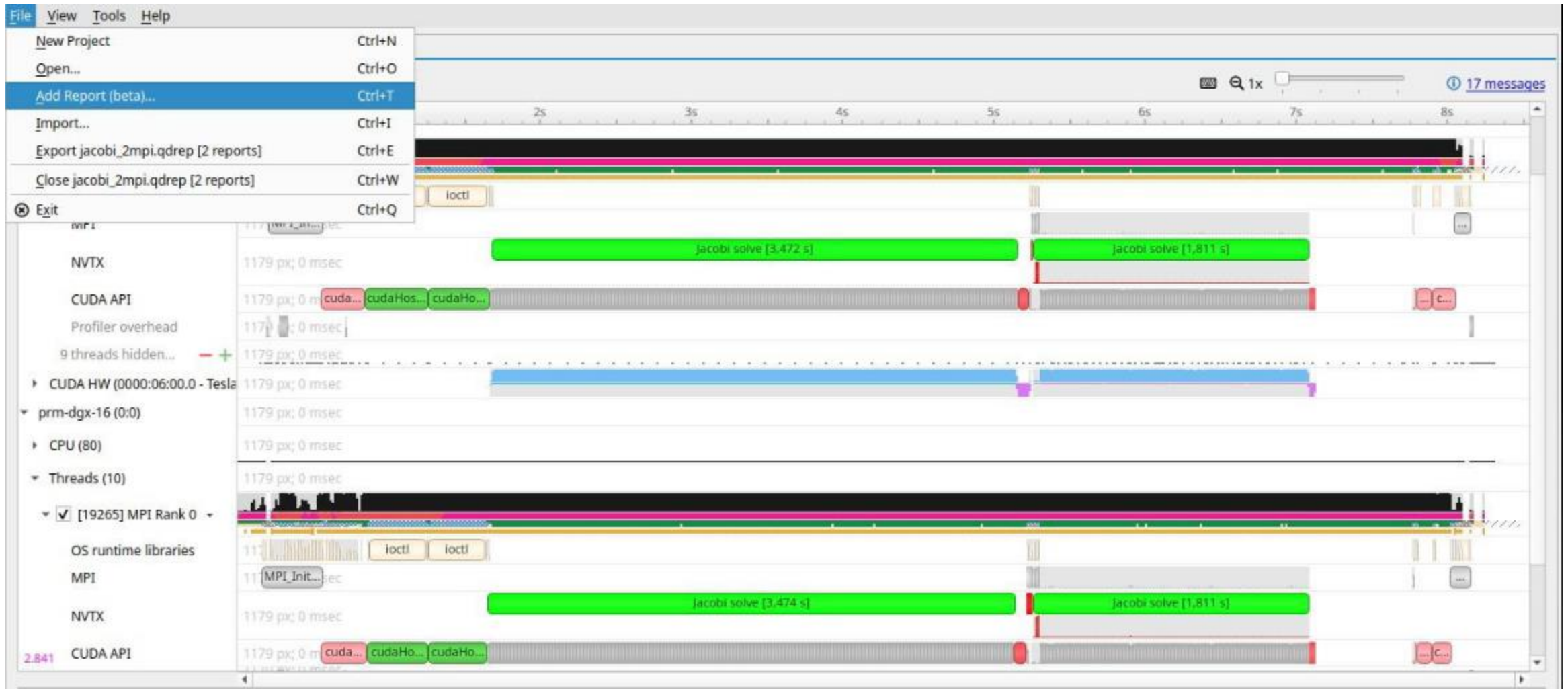


OPENMP



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

The screenshot displays the NVIDIA NSIGHT Compute interface. The top section shows a summary table of kernel launches. The bottom section provides a detailed breakdown of GPU metrics, including throughput, utilization, and workload analysis.

Page:	Summary	Launch:	0 - 43843 - device_tea_leaf_ppcg_sol	▼	▼	Add Baseline	▼	Apply Rules	▼	Occupancy Calculator	Copy as Image
	Launch	Time	Cycles	Regs	GPU	SM Frequency	CC	Process			
Current	43843 - device_tea_leaf_ppcg_solve_init (126, 1001, 1)x(32, 4, 1)	217.63 usecond	297,114	40	0 - NVIDIA GeForce RTX 2080 Ti	1.36 cycle/nsecond	7.5	[15958] tea_leaf			

ID	Time	API Call ID	Function Name	Demangled Name	Process	Device Name	Grid Size	Block Size	Cycles [cycle]	Duration [msecond]	Compute Throughput [%]	Memc
0	2021-Dec-1...	43843	device_tea_leaf_ppcg...	device_te...	[15958] tea_leaf	NVIDIA GeForce...	126, 1001, 1	32, 4, 1	297,114	0.22	77.89	
1	2021-Dec-1...	43857	device_tea_leaf_ppcg_sol...	device_tea_J...	[15958] tea_leaf	NVIDIA GeForce RT...	126, 1001, 1	32, 4, 1	1,264,921	0.94	54.78	
2	2021-Dec-1...	43860	device_tea_leaf_ppcg_sol...	device_tea_J...	[15958] tea_leaf	NVIDIA GeForce RT...	126, 1001, 1	32, 4, 1	1,462,446	1.07	86.22	
3	2021-Dec-1...	43863	device_tea_leaf_ppcg_sol...	device_tea_J...	[15958] tea_leaf	NVIDIA GeForce RT...	126, 1001, 1	32, 4, 1	1,443,836	1.06	23.81	

Page:	Details	Launch:	0 - 43843 - device_tea_leaf_ppcg_sol	▼	▼	Add Baseline	▼	Apply Rules	▼	Occupancy Calculator	Copy as Image
	Launch	Time	Cycles	Regs	GPU	SM Frequency	CC	Process			
Current	43843 - device_tea_leaf_ppcg_solve_init (126, 1001, 1)x(32, 4, 1)	217.63 usecond	297,114	40	0 - NVIDIA GeForce RTX 2080 Ti	1.36 cycle/nsecond	7.5	[15958] tea_leaf			

GPU Speed Of Light Throughput		
Compute (SM) Throughput [%]	77.89	Duration [usecond]
Memory Throughput [%]	45.03	Elapsed Cycles [cycle]
L1/TEX Cache Throughput [%]	68.22	SM Active Cycles [cycle]
L2 Cache Throughput [%]	2.30	SM Frequency [cycle/nsecond]
DRAM Throughput [%]	0.12	DRAM Frequency [cycle/nsecond]

High Compute Throughput		
Compute is more heavily utilized than Memory: Look at the Compute Workload Analysis report section to see what the compute pipelines are spending their time doing. Also, consider whether any computation is redundant and could be reduced or moved to look-up tables.		

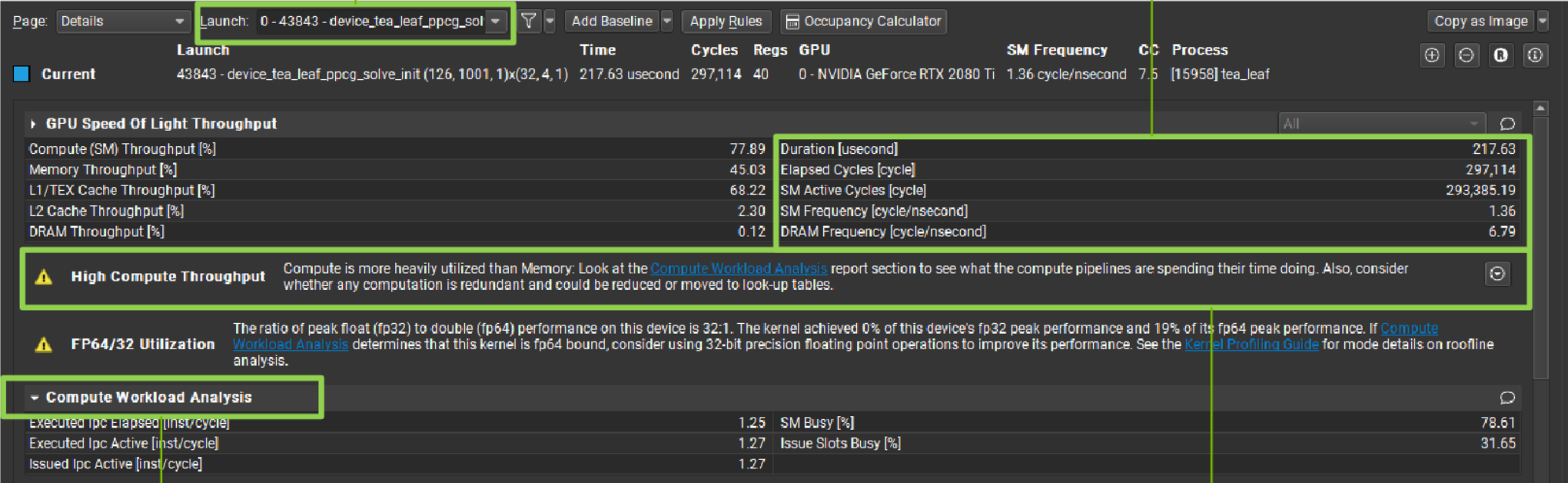
FP64/32 Utilization		
The ratio of peak float (fp32) to double (fp64) performance on this device is 32:1. The kernel achieved 0% of this device's fp32 peak performance and 19% of its fp64 peak performance. If Compute Workload Analysis determines that this kernel is fp64 bound, consider using 32-bit precision floating point operations to improve its performance. See the Kernel Profiling Guide for more details on runtime analysis.		

Compute Workload Analysis		
Executed lpc Elapsed [inst/cycle]	1.25	SM Busy [%]
Executed lpc Active [inst/cycle]	1.27	Issue Slots Busy [%]
Issued lpc Active [inst/cycle]	1.27	

PROFILER REPORT

Selected result

Metric values

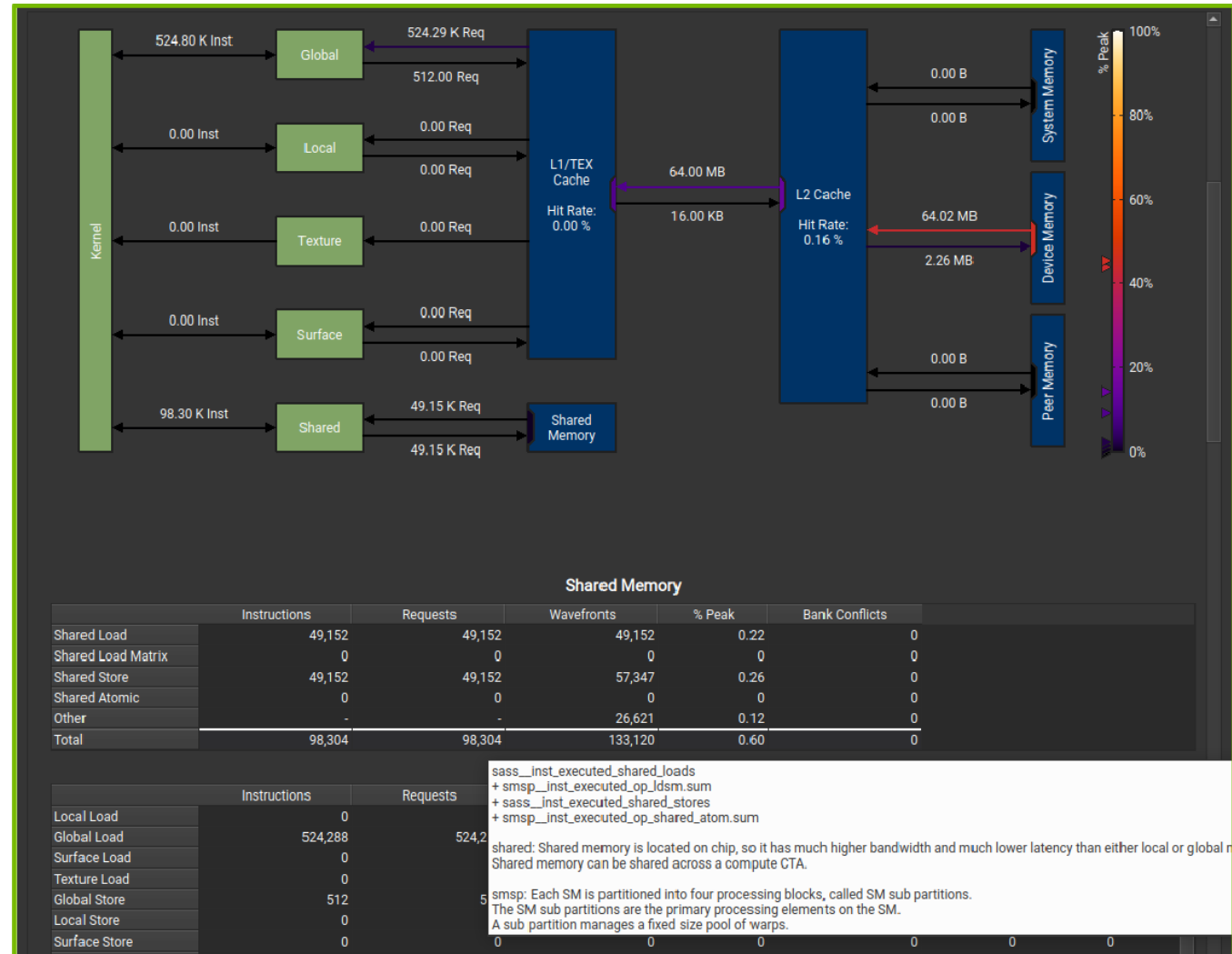


Expandable
Sections

Expert Analysis
(Rules)

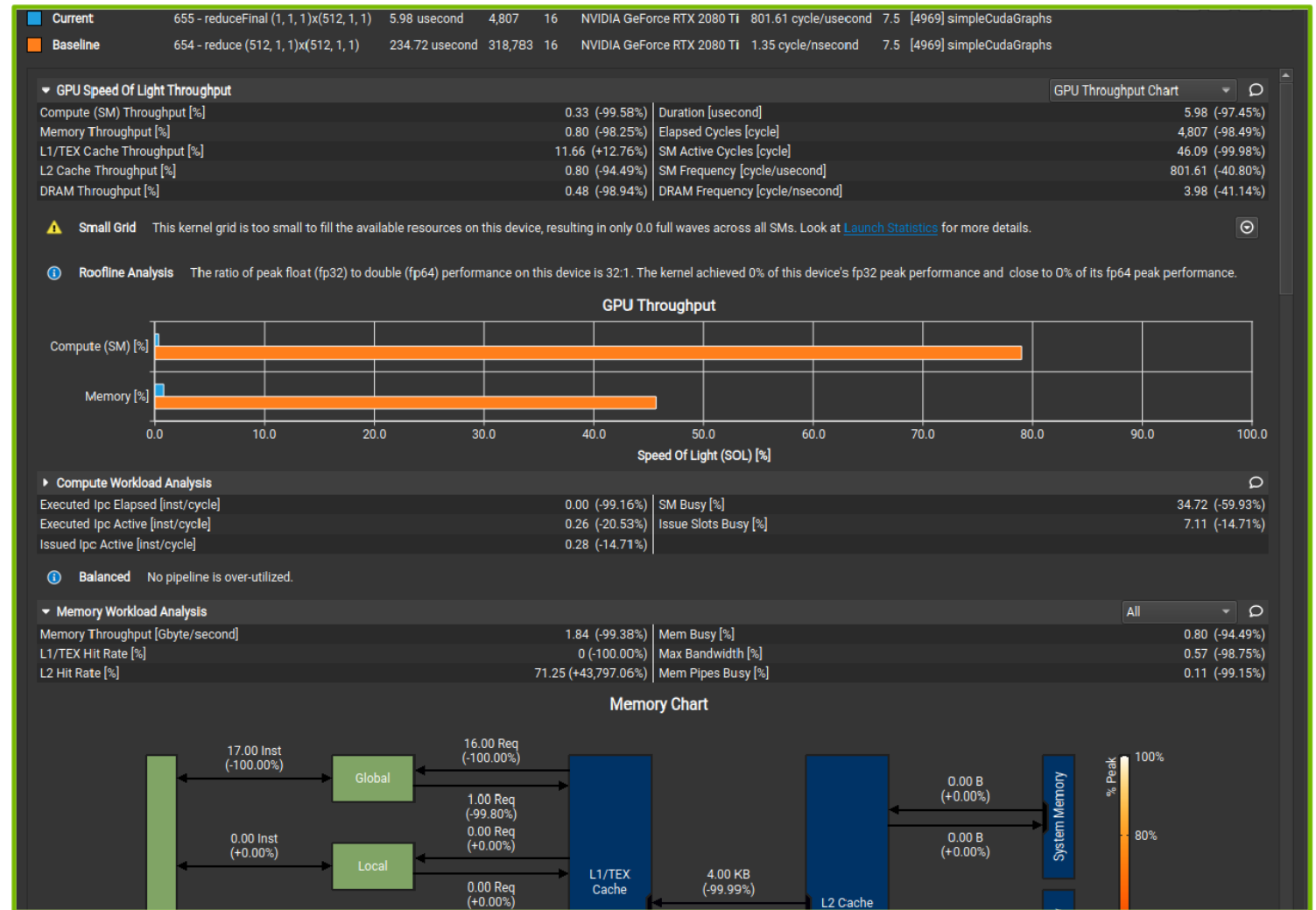
DATA TRANSFER ANALYSIS

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



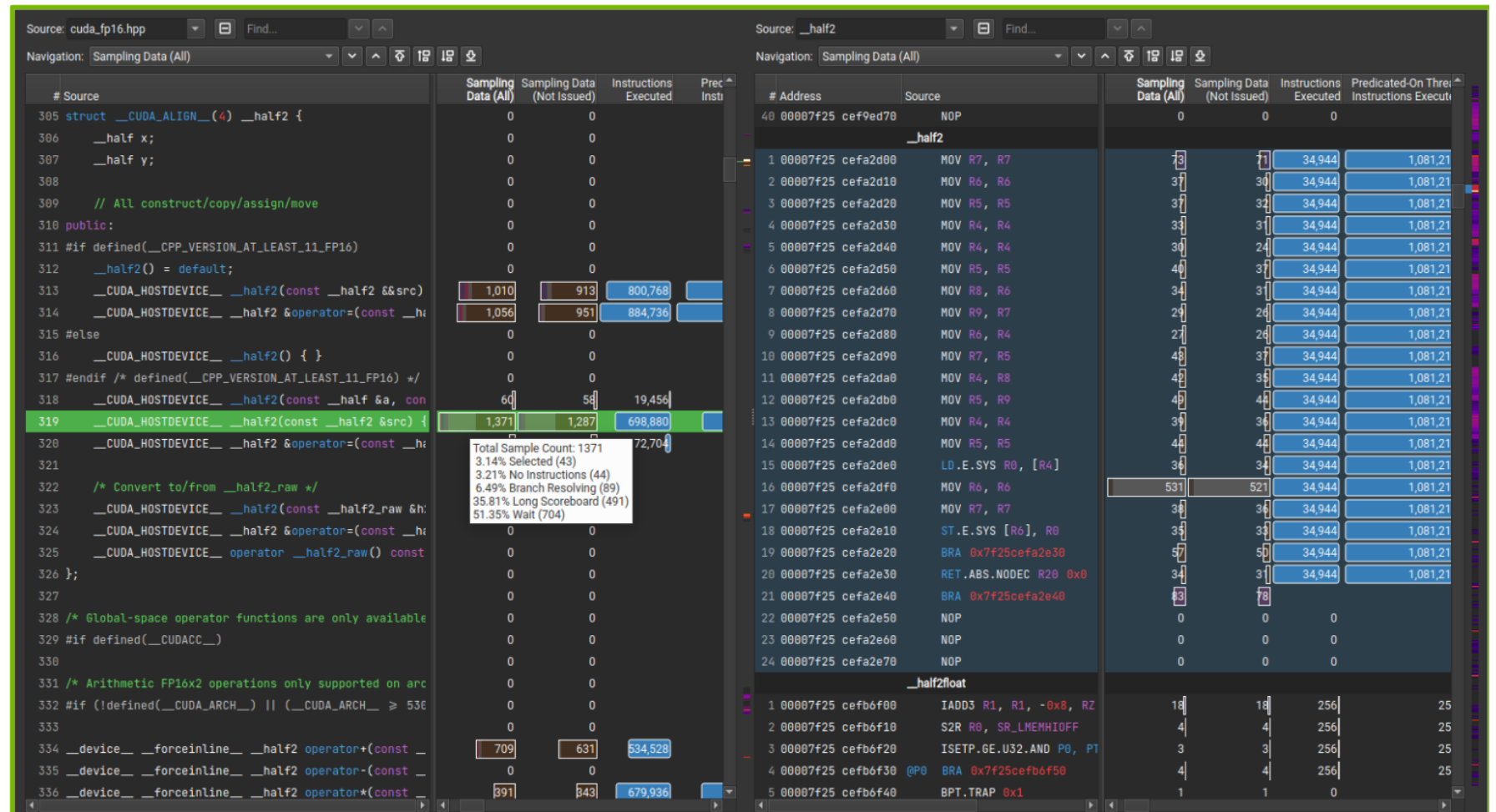
BASELINE COMPARISON

- 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 resources are wasted

Compute Capability: 7.5

Threads Per Block: 128

Shared Memory Size Config (bytes): 65536

Registers Per Thread: 40

CUDA version: 11.0

Shared Memory Per Block (bytes): 3072

Global Load Cache Mode: L1+L2 (ca)

Apply

Reset

Tables

Graphs

GPU Data

Occupancy Data:

Property	Value
Active Threads per Multiprocessor	1024
Active Warps per Multiprocessor	32
Active Thread Blocks per Multiprocessor	8
Occupancy of each Multiprocessor	100 %

Physical Limit of GPU (7.5):

Property	Limit
Threads per Warp	32
Max Warps per Multiprocessor	32
Max Thread Blocks per Multiprocessor	16
Max Threads per Multiprocessor	1024
Maximum Thread Block Size	1024
Registers per Multiprocessor	65536
Max Registers per Thread Block	65536
Max Registers per Thread	255
Shared Memory per Multiprocessor (bytes)	65536
Max Shared Memory per Block	65536
Register Allocation Unit Size	256
Register Allocation Granularity	warp
Shared Memory Allocation Unit Size	256
Warp Allocation Granularity	4
Shared Memory Per Block (bytes) (CUDA runtime use)	0

Allocated Resources:

Resources	Per Block	Limit Per SM	Allocatable Blocks Per SM
Warps (Threads Per Block / Threads Per Warp)	4	32	8
Registers (Warp limit per SM due to per-warp reg count)	4	48	12
Shared Memory (Bytes)	3072	65536	21

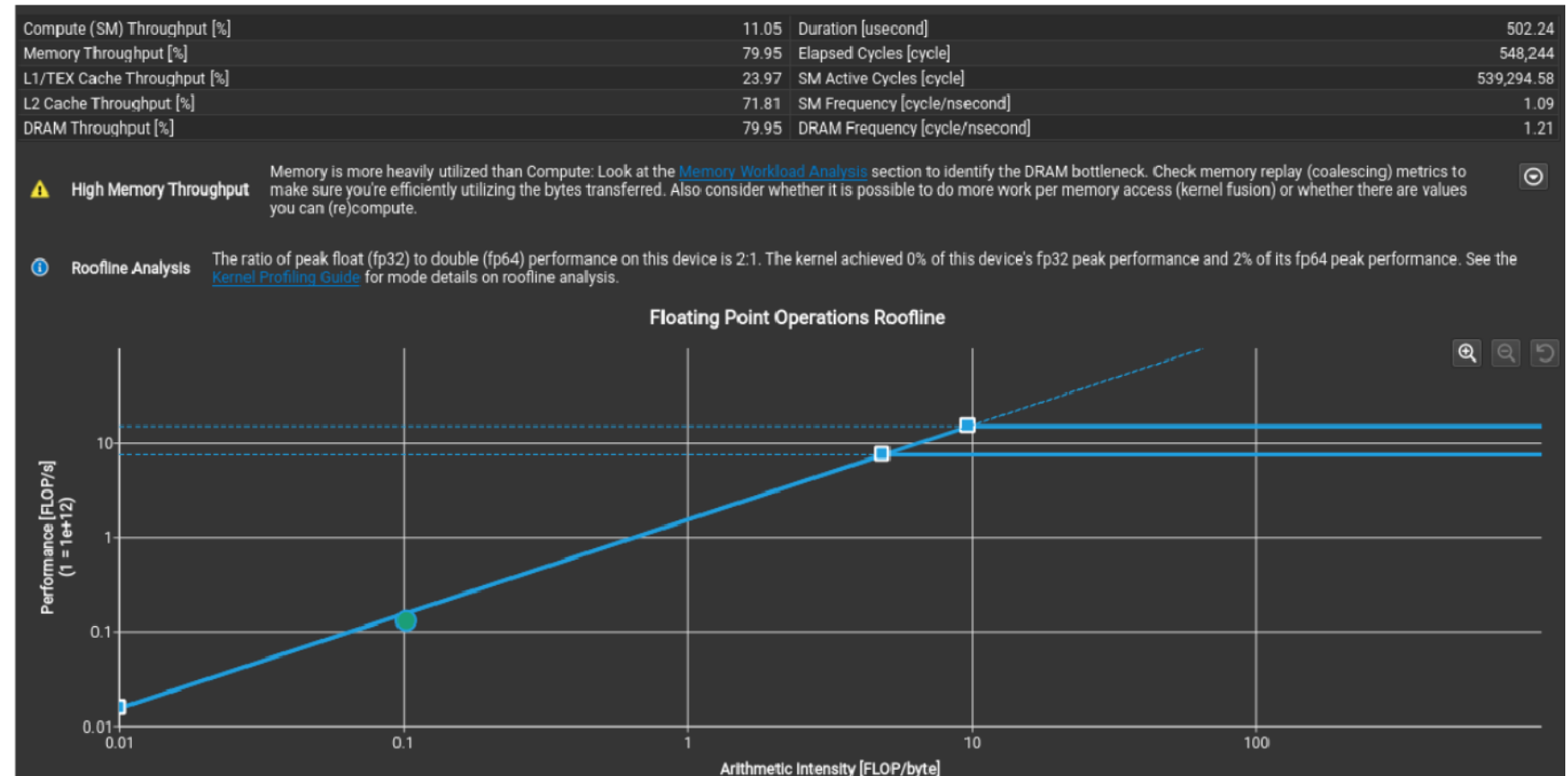
Occupancy Limiters:

Limited By	Blocks per SM	Warps Per Block	Warps Per SM
Max Warps or Max Blocks per Multiprocessor	8	4	32
Registers per Multiprocessor	12		
Shared Memory per Multiprocessor	21		

The occupancy is limited by block size

ROOFLINE ANALYSIS

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



QUESTIONS?



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



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

