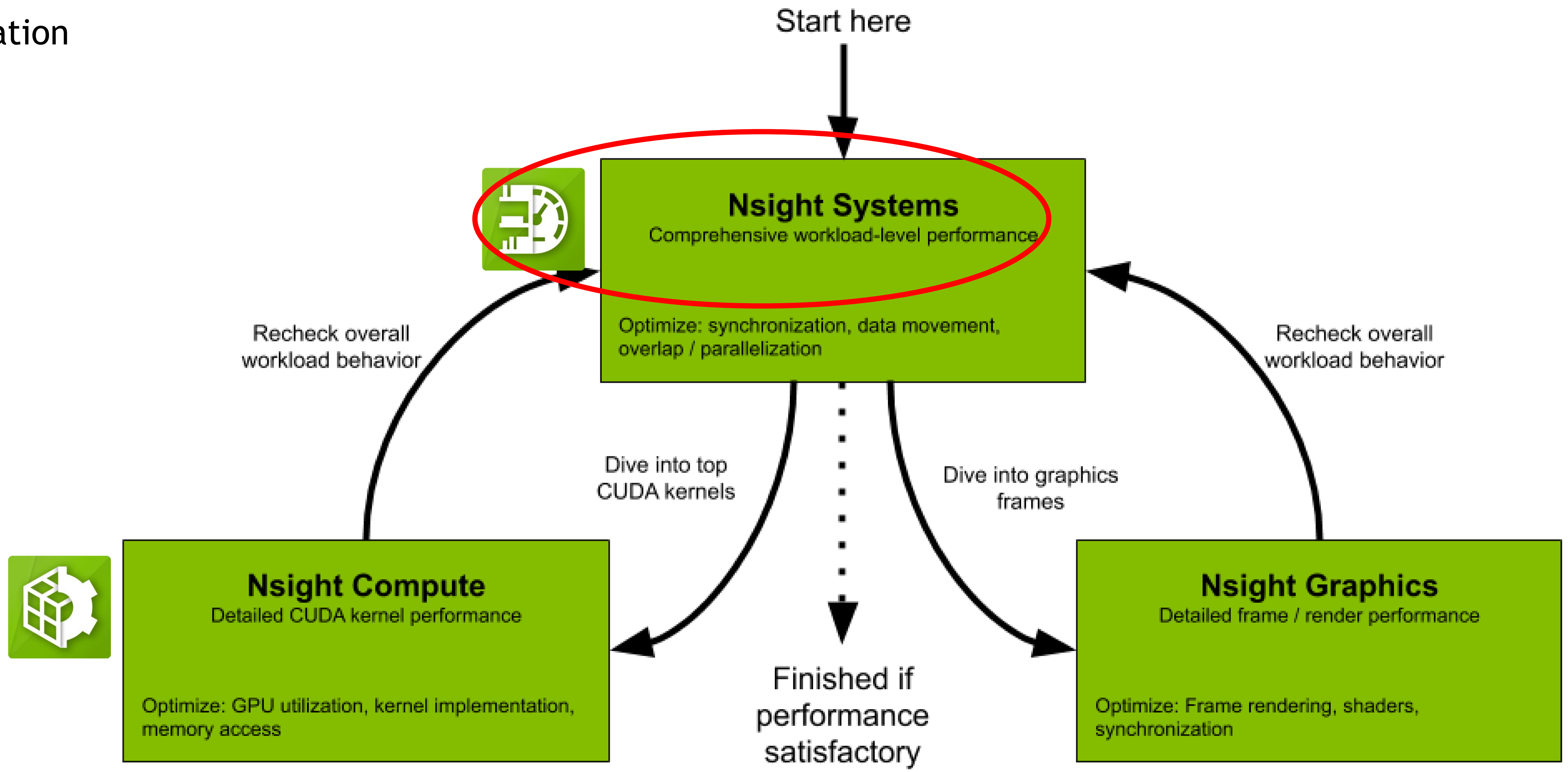
Profiling on Grace Hopper

Jiri Kraus, Principal Devtech Compute | CASA Workshop JSC/Oct. 2024

The Nsight Suite Components

How the pieces fit together

- Nsight **Systems**: Coarse-grained, whole-application
- Nsight **Compute**: Fine-grained, kernel-level
- NVTX: Support and structure across tools
- Main purpose: Performance optimization
  - But at their core, advanced measurement tools



Deployment and Setup

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

- Multiple deployed versions may already exist, as profilers come…
  - …with the CUDA Toolkit
  - …with HPC SDK
  - …standalone
- Recommendation: Always install latest versions, especially for collection
- All Nsight profilers available standalone - https://developer.nvidia.com/nsight-systems/get-started
  - Via Website or Repositories, package managers (https://developer.download.nvidia.com/devtools/repos/)
- Compatibility: https://developer.nvidia.com/nsight-systems/get-started#platforms
  - Collect and Analyze via GUI
  - Collect on CLI, Analyze via GUI
  - Analysis GUI version >= collector version

Nsight Systems on JEDI

- Multiple deployed versions may already exist, as profilers come…
  - …with the CUDA Toolkit (2023.2.3 with CUDA 12 on JEDI):

```
[kraus1@jpblt-s01-01 ~]$ module load CUDA
[kraus1@jpblt-s01-01 ~]$ which nsys
/p/software/jedi/stages/2024/software/CUDA/12/bin/nsys
[kraus1@jpblt-s01-01 ~]$ nsys --version
NVIDIA Nsight Systems version 2023.2.3.1001-32894139v0
```

  - …with HPC SDK (2024.1.1 with 24.3 on JEDI):

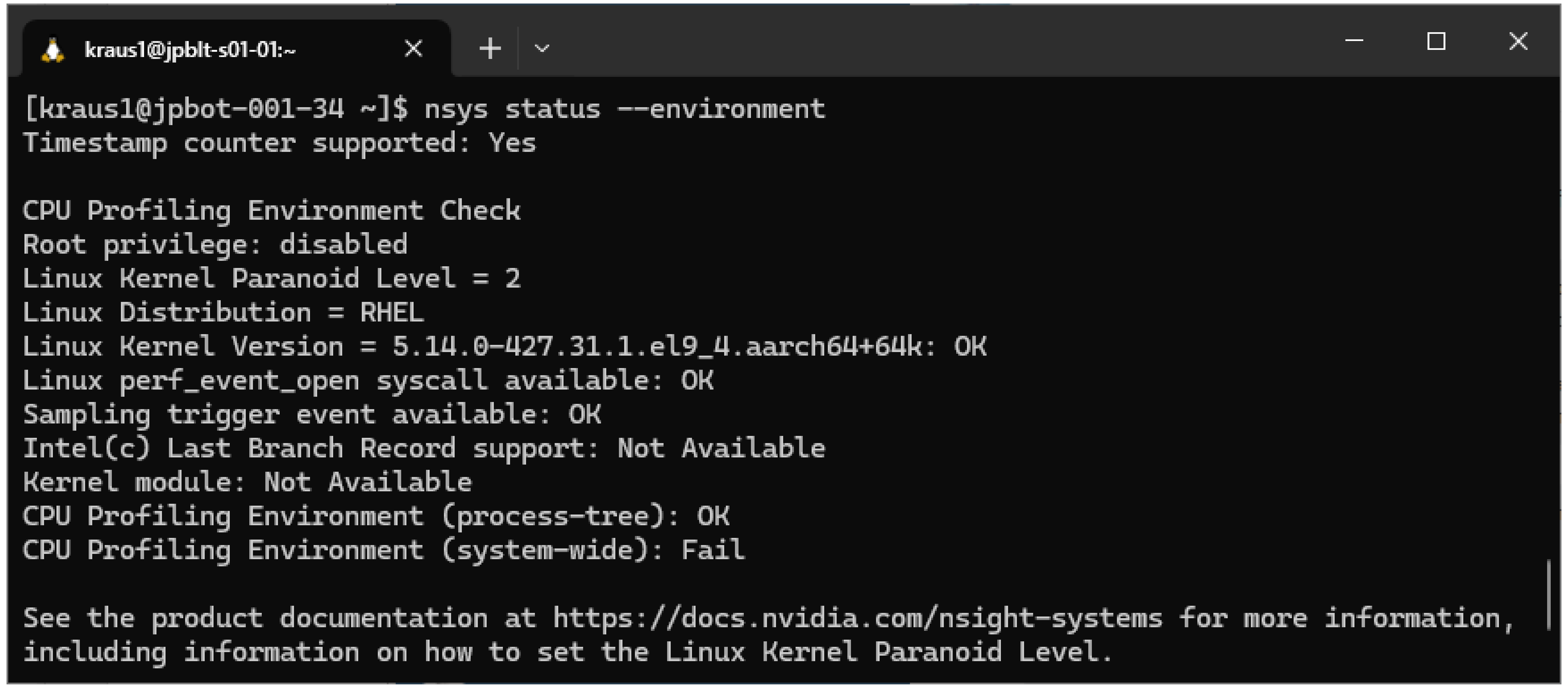
```
[kraus1@jpblt-s01-01 ~]$ module load NVHPC/24.3-CUDA-12
[kraus1@jpblt-s01-01 ~]$ which nsys
/p/software/jedi/stages/2024/software/NVHPC/24.3-CUDA-12/Linux_aarch64/24.3/compilers/bin/nsys
[kraus1@jpblt-s01-01 ~]$ nsys --version
NVIDIA Nsight Systems version 2024.1.1.59-241133802077v0
```

  - …standalone (2024.4.1 on JEDI) – latest available 2024.6.1

```
[kraus1@jpblt-s01-01 ~]$ module load Nsight-Systems/2024.4.1
[kraus1@jpblt-s01-01 ~]$ which nsys
/p/software/jedi/stages/2024/software/Nsight-Systems/2024.4.1-GCCcore-12.3.0/bin/nsys
[kraus1@jpblt-s01-01 ~]$ nsys --version
NVIDIA Nsight Systems version 2024.4.1.61-244134315967v0
```

Call Stack, CPU Metric, and GPU Metric Sampling

- Linux kernel setting required for call stack sampling (`Linux Kernel Paranoid Level <= 2`) and CPU metric sampling (`Linux Kernel Paranoid Level <= 0`):
  - `sudo sh -c 'echo kernel.perf_event_paranoid=2|0 > /etc/sysctl.d/local.conf'`
  - https://docs.nvidia.com/nsight-systems/InstallationGuide/index.html#requirements-for-x86-64-power-and-arm-sbsa-targets-on-linux
  - `Linux Kernel Paranoid Level = 2` default on JUWELS-Booster and JEDI
  - Setting `Linux Kernel Paranoid Level = 0` possible on JUWELS-Booster with slurm option: `--disable-perfparanoid Setting Kernel not to be paranoid with respect to perf`
    - JEDI: Support will be added once ParTec software stack is integrated. If there is an urgent need contact jedi-adm@fz-juelich.de
- Some DCGM monitoring can interfere with GPU Metric Sampling, see https://docs.nvidia.com/datacenter/dcgm/latest/user-guide/feature-overview.html#concurrent-usage-of-nvidia-profiling-tools
  - DCGM and CUPTI are using the same hardware profiling capabilities
  - JUWELS-Booster slurm option: `--disable-dcgm Disable DCGM Metrics collection. This is required for using profiling tools on the GPUs`
    - JEDI: DCGM not running yet so not needed



A First (I)Nsight

Recording with the CLI

- Load Latest Nsight Systems
  - `module load Nsight-Systems/2024.4.1`
- Use the command line
  - `srun nsys profile --trace=cuda,nvtx,mpi --force-overwrite=true --output=my_report.%q{SLURM_PROCID} \ ./jacobi -niter 10`
- Inspect results: Open the report file in the GUI
  - Also possible to get details on command line
  - Either add `--stats` to profile command line, or: `nsys stats --help`

- Runs set of reports on command line, customizable (**sqlite** + `Python`):
  - Useful to check validity of profile, identify important kernels

```
Running [.../reports/gpukernsum.py jacobi_metrics_more-nvtx.0.sqlite]...

Time(%)  Total Time (ns)  Instances  Avg (ns)   Med (ns)   Min (ns)  Max (ns)  StdDev (ns)           Name
-------  ---------------  ---------  ---------  ---------  --------  --------  -----------  ---------------------
   99.9         36750359         20  1837518.0  1838466.5    622945   3055044    1245121.7  void jacobi_kernel
    0.1            22816          2    11408.0    11408.0      7520     15296       5498.5  initialize_boundaries
```

Example: Available reports

Extending and customizing

- `nsys stats --help-reports`
  - list all available reports
- Example: CUDA API sum, customize to show only “Stream” APIs
  - `cp /opt/nvidia/nsight-systems/2024.1.1/host-linux-x64/reports/cuda_api_sum.py my_cuda_sum.py`
  - Editing `my_cuda_sum.py`, for example:

```
…
LEFT JOIN
    StringIds AS ids
    ON ids.id == summary.nameId
WHERE Name LIKE "%Stream%”
ORDER BY 2 DESC
…
```

- Running in the same way (reports from current folder picked up):
  - `nsys stats -r my_cuda_sum --timeunit ms jacobi_2024.1-0.nsys-rep`

```
Processing [jacobi_2024.1-0.sqlite] with [./my_cuda_sum.py]...

 ** CUDA API Summary (my_cuda_sum):
```

Time (%)	Total Time (ms)	Num Calls	Avg (ms)	Med (ms)	Min (ms)	Max (ms)	StdDev (ms)	Name
1.5	38.1181	40	0.9530	0.3449	0.0023	3.1075	1.2879	cudaStreamSynchronize
0.0	0.4594	18	0.0255	0.0020	0.0018	0.4024	0.0941	cuStreamCreate
0.0	0.0925	80	0.0012	0.0009	0.0006	0.0097	0.0012	cudaStreamWaitEvent
0.0	0.0646	6	0.0108	0.0064	0.0043	0.0294	0.0098	cudaStreamDestroy
0.0	0.0492	6	0.0082	0.0026	0.0024	0.0343	0.0128	cudaStreamCreate
0.0	0.0457	2	0.0229	0.0229	0.0113	0.0344	0.0163	cuStreamDestroy_v2
0.0	0.0101	4	0.0025	0.0025	0.0022	0.0029	0.0004	cuStreamSynchronize

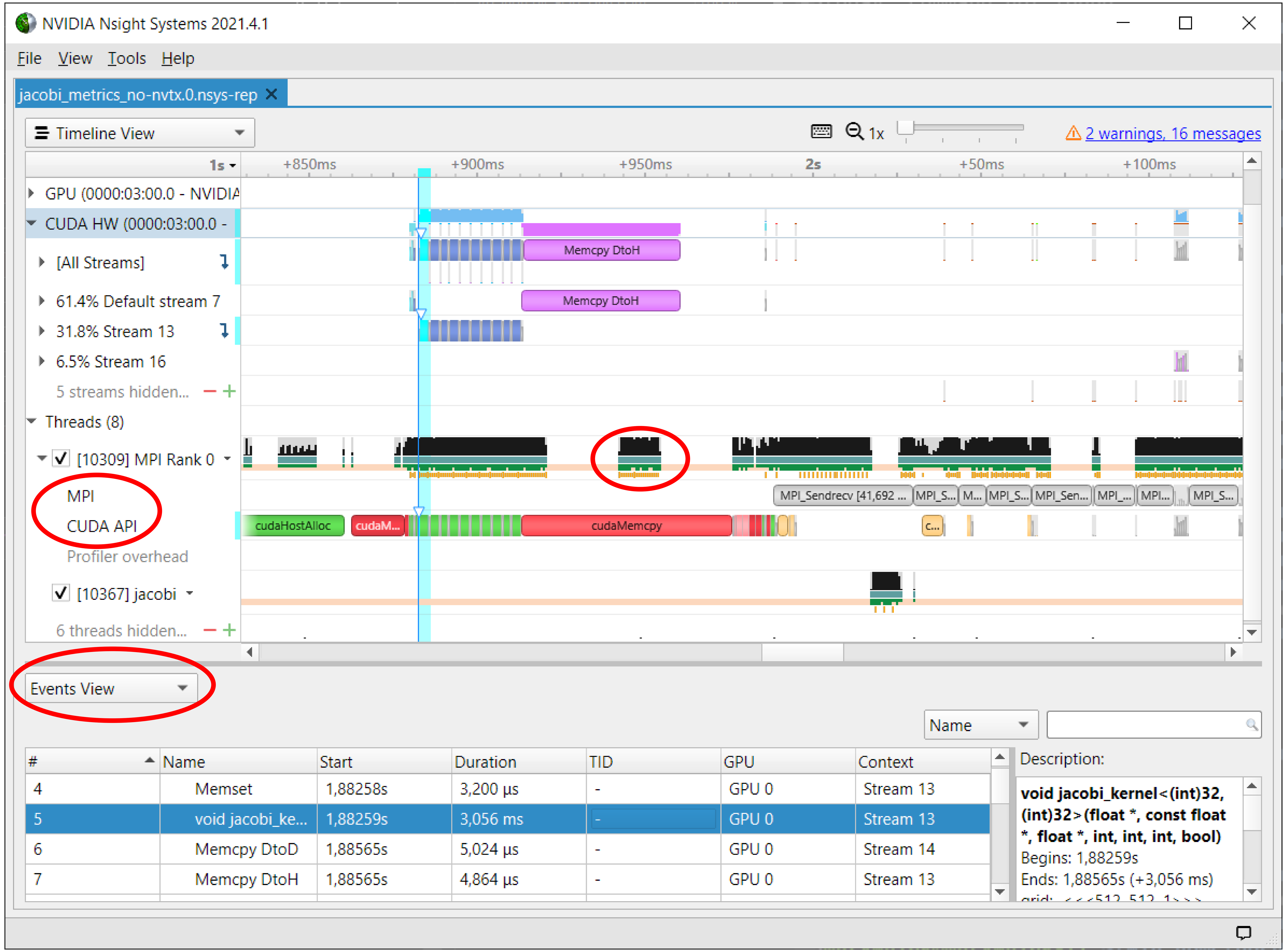
```
The following built-in reports are available:

  cuda_api_gpu_sum[:nvtx-name][:base|:mangled] -- CUDA Summary (API/Kernels/MemOps)
  cuda_api_sum -- CUDA API Summary
  cuda_api_trace -- CUDA API Trace
  cuda_gpu_kern_gb_sum[:nvtx-name][:base|:mangled] -- CUDA GPU Kernel/Grid/Block Summary
  cuda_gpu_kern_sum[:nvtx-name][:base|:mangled] -- CUDA GPU Kernel Summary
  cuda_gpu_mem_size_sum -- CUDA GPU MemOps Summary (by Size)
  cuda_gpu_mem_time_sum -- CUDA GPU MemOps Summary (by Time)
  cuda_gpu_sum[:nvtx-name][:base|:mangled] -- CUDA GPU Summary (Kernels/MemOps)
  cuda_gpu_trace[:nvtx-name][:base|:mangled] -- CUDA GPU Trace
  cuda_kern_exec_sum[:nvtx-name][:base|:mangled] -- CUDA Kernel Launch & Exec Time Summary
  cuda_kern_exec_trace[:nvtx-name][:base|:mangled] -- CUDA Kernel Launch & Exec Time Trace
  dx11_pix_sum -- DX11 PIX Range Summary
  dx12_gpu_marker_sum -- DX12 GPU Command List PIX Ranges Summary
  dx12_pix_sum -- DX12 PIX Range Summary
  mpi_event_sum -- MPI Event Summary
  mpi_event_trace -- MPI Event Trace
  network_congestion[:ticks_threshold=<ticks_per_ms>] -- Network Devices Congestion
  nvtx_gpu_proj_sum -- NVTX GPU Projection Summary
  nvtx_gpu_proj_trace -- NVTX GPU Projection Trace
  nvtx_kern_sum[:base|:mangled] -- NVTX Range Kernel Summary
  nvtx_pushpop_sum -- NVTX Push/Pop Range Summary
  nvtx_pushpop_trace -- NVTX Push/Pop Range Trace
  nvtx_startend_sum -- NVTX Start/End Range Summary
  nvtx_sum -- NVTX Range Summary
  nvvideo_api_sum -- NvVideo API Summary
  openacc_sum -- OpenACC Summary
  opengl_khr_gpu_range_sum -- OpenGL KHR_debug GPU Range Summary
  opengl_khr_range_sum -- OpenGL KHR_debug Range Summary
  openmp_sum -- OpenMP Summary
  osrt_sum -- OS Runtime Summary
  um_cpu_page_faults_sum -- Unified Memory CPU Page Faults Summary
  um_sum[:rows=<limit>] -- Unified Memory Analysis Summary
  um_total_sum -- Unified Memory Totals Summary
  vulkan_api_sum -- Vulkan API Summary
  vulkan_api_trace -- Vulkan API Trace
  vulkan_gpu_marker_sum -- Vulkan GPU Range Summary
  vulkan_marker_sum -- Vulkan Range Summary
  wddm_queue_sum -- WDDM Queue Utilization Summary

For more information, use '--help-reports <report_name>'
```

System-level Profiling with Nsight Systems

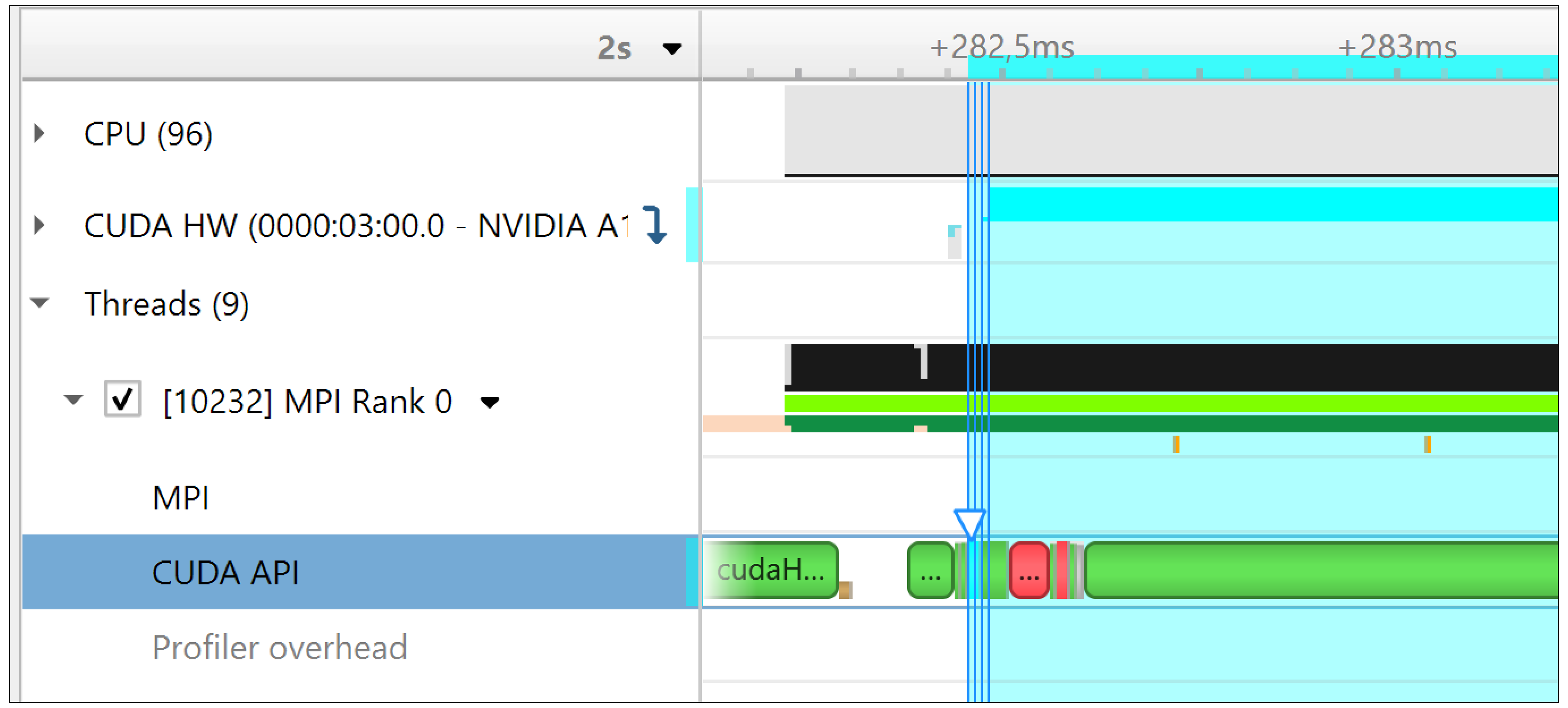
- Global timeline view
  - CUDA HW: streams, kernels, memory
- Different traces, e.g. CUDA, MPI
  - correlations API <-> HW
- Stack samples
  - bottom-up, top-down for CPU code
- (GPU metrics)
- Events View
- looks at single process (tree)

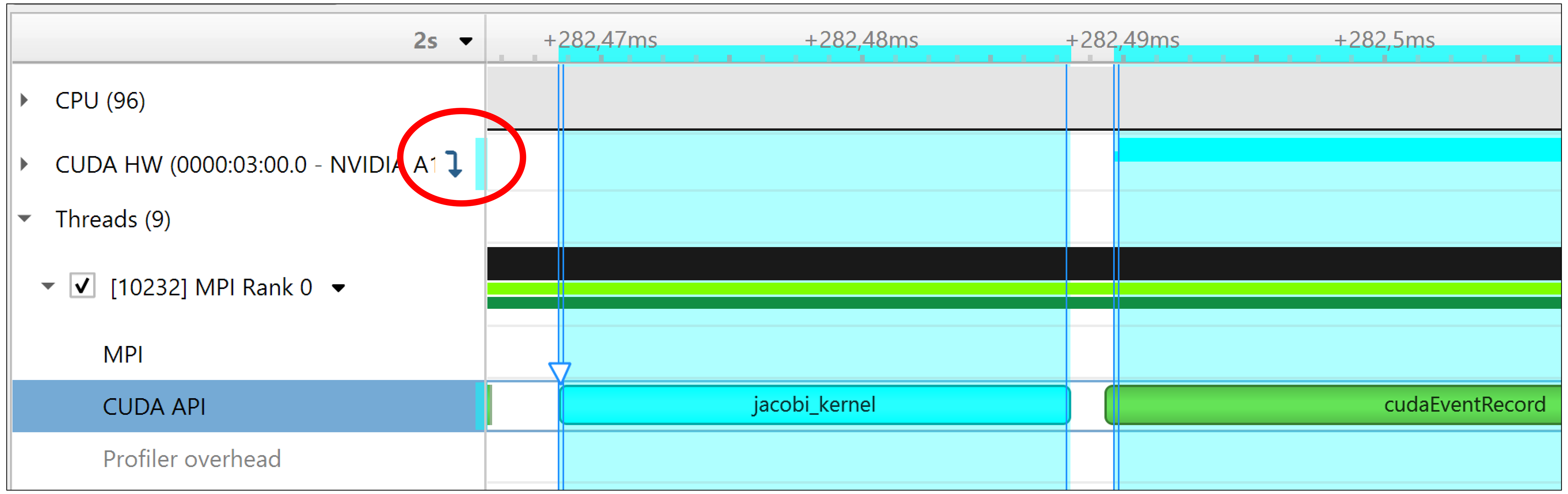


#	Name	Start	Duration	TID	GPU	Context
4	Memset	1,88258s	3,200 µs	-	GPU 0	Stream 13
5	void jacobi_ke...	1,88259s	3,056 ms	-	GPU 0	Stream 13
6	Memcpy DtoD	1,88565s	5,024 µs	-	GPU 0	Stream 14
7	Memcpy DtoH	1,88565s	4,864 µs	-	GPU 0	Stream 13

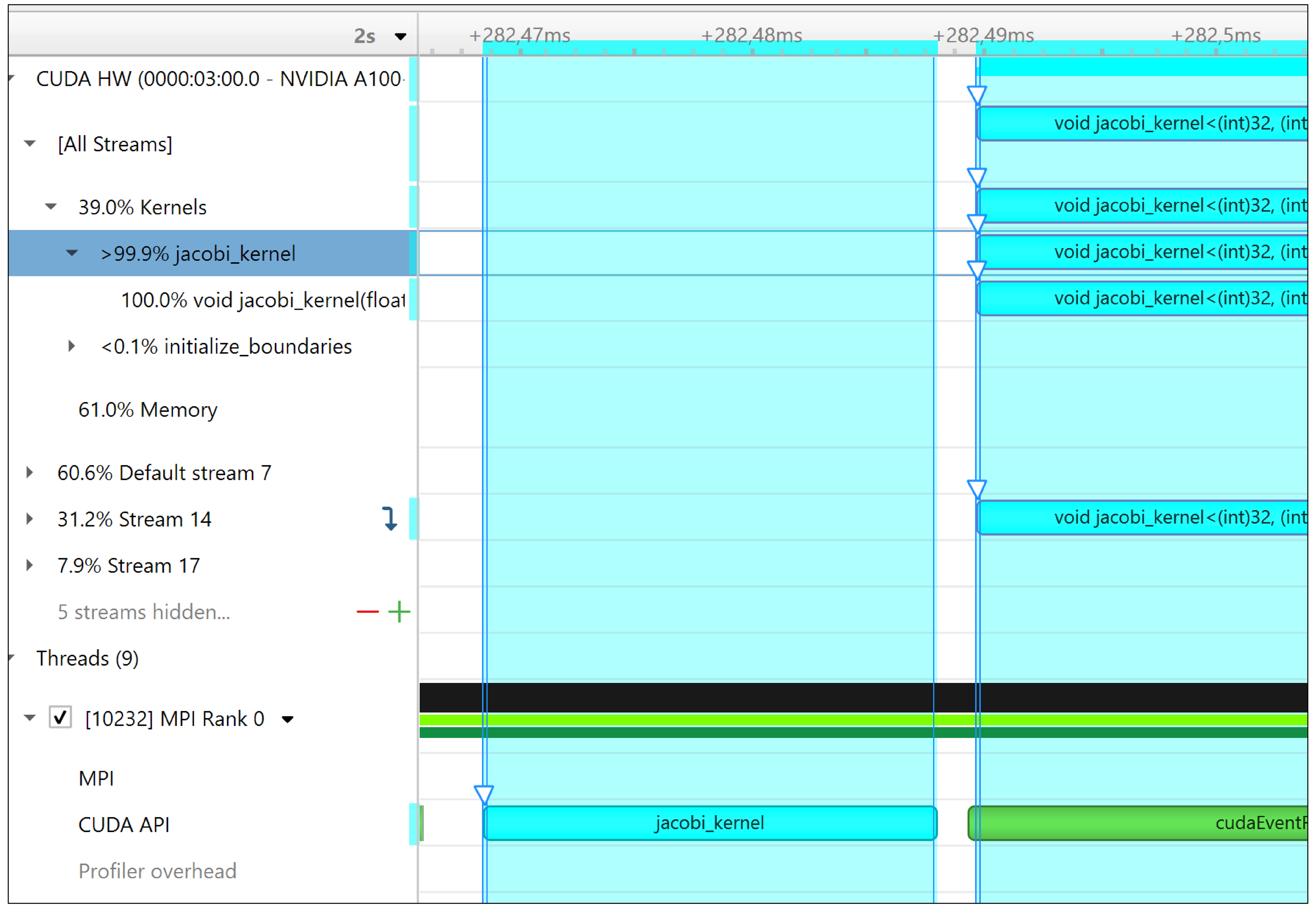
Correlating Events on the Timeline

Selecting events in one row highlights related events





- CUDA's execution model is asynchronous
  - Kernel launch on host returns
  - Kernel runs on GPU
- Visualized in profiler

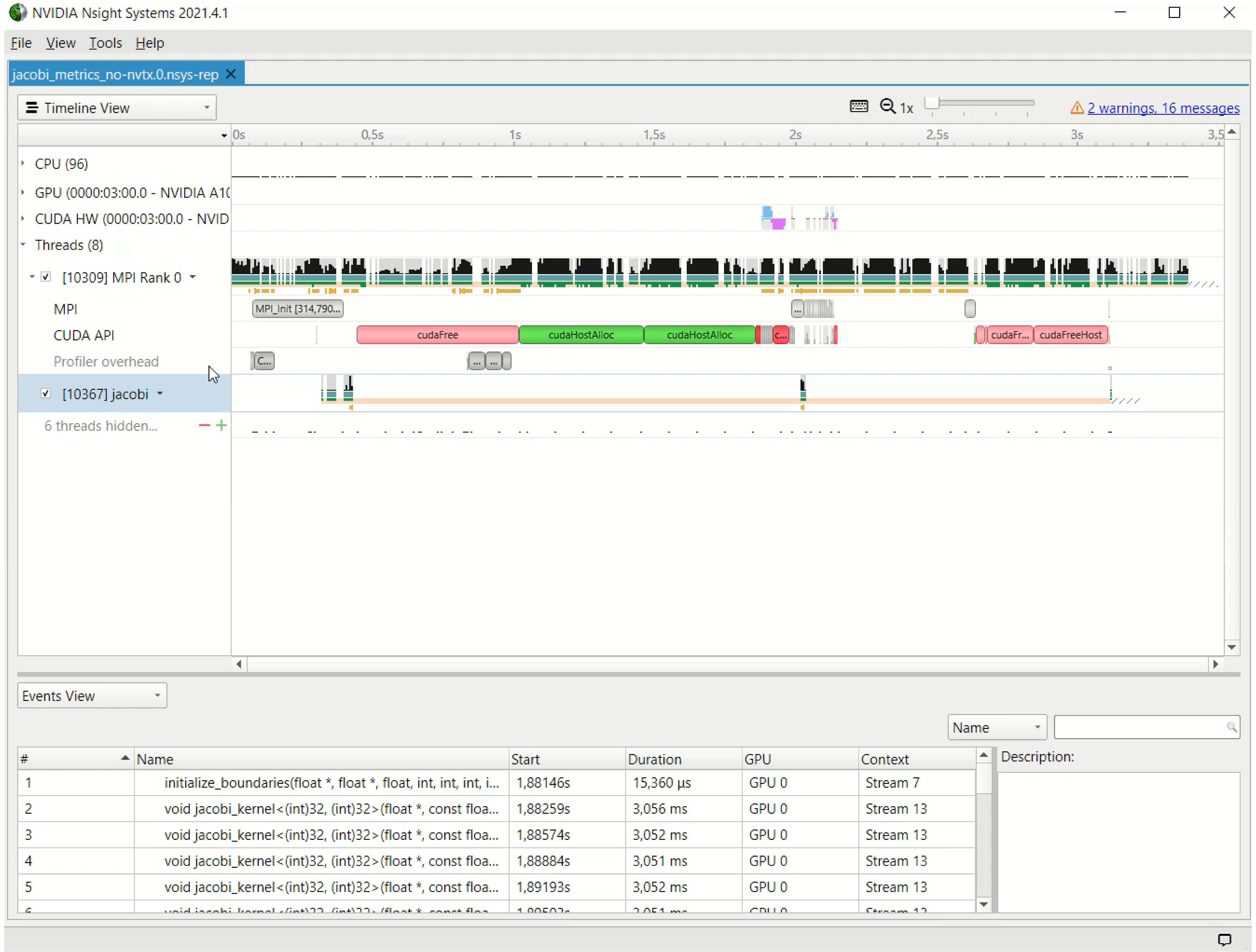


Correlating Events on the Timeline

Selecting events in one row highlights related events

Nsight Systems Basic Workflow

Navigating the timeline and finding interesting areas



#	Name	Start	Duration	GPU	Context
1	initialize_boundaries(float *, float *, float, int, int, int, i...	1,88146s	15,360 µs	GPU 0	Stream 7
2	void jacobi_kernel<(int)32, (int)32>(float *, const floa...	1,88259s	3,056 ms	GPU 0	Stream 13
3	void jacobi_kernel<(int)32, (int)32>(float *, const floa...	1,88574s	3,052 ms	GPU 0	Stream 13
4	void jacobi_kernel<(int)32, (int)32>(float *, const floa...	1,88884s	3,051 ms	GPU 0	Stream 13
5	void jacobi_kernel<(int)32, (int)32>(float *, const floa...	1,89193s	3,052 ms	GPU 0	Stream 13

Adding NVTX

Simple range-based API

- `#include "nvtx3/nvToolsExt.h"`
  - NVTX v3 is header-only, needs just `-ldl`
  - C++ and Python APIs
- Fortran: NVHPC compilers include module
  - Just `use nvtx` and `-lnvhpcwrapnvtx`
  - Other compilers: See blog posts linked below
- Definitely: Include `PUSH/POP` macros (see links below)

  `PUSH_RANGE(`*`name, color_idx`*`)`
- Sprinkle them strategically through code
  - Use hierarchically: Nest ranges
- Not shown: Advanced usage (domains, …)
- Similar range-based annotations exist for other tools
  - e.g. SCOREP_USER_REGION_BEGIN

```
int main(int argc, char** argv) {
    PUSH_RANGE("main", 0)
    PUSH_RANGE("init", 1)
    do_initialization();
    POP_RANGE
    /* ... */
    PUSH_RANGE("computation", 2)
    jacobi_kernel<<</* ... */, compute_stream>>>(...);
    cudaStreamSynchronize(compute_stream);
    POP_RANGE
    /* ... */
    POP_RANGE
}
```

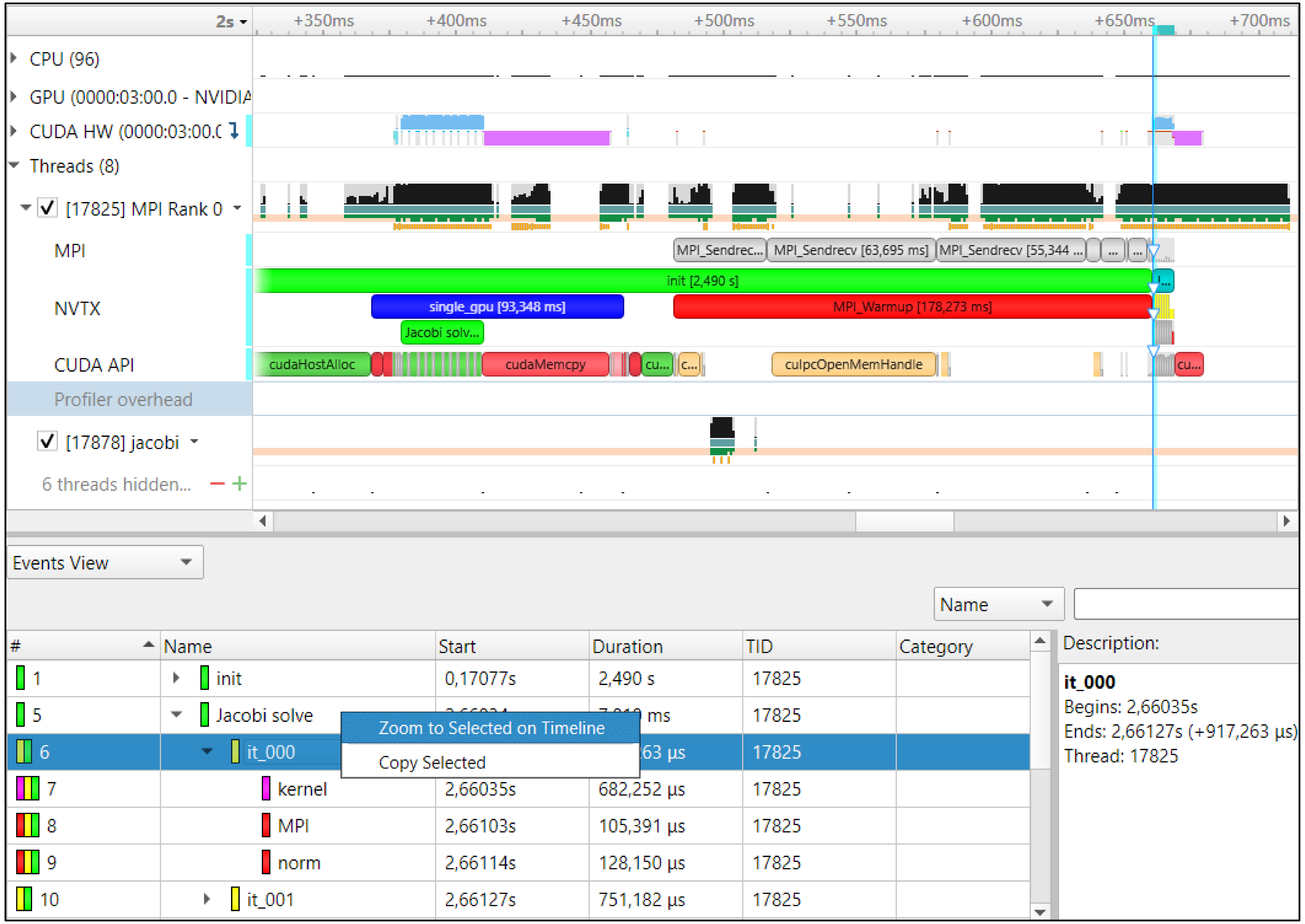
https://github.com/NVIDIA/NVTX and https://nvidia.github.io/NVTX/#how-do-i-use-nvtx-in-my-code

https://developer.nvidia.com/blog/cuda-pro-tip-generate-custom-application-profile-timelines-nvtx/
https://developer.nvidia.com/blog/customize-cuda-fortran-profiling-nvtx/

Adding some Color

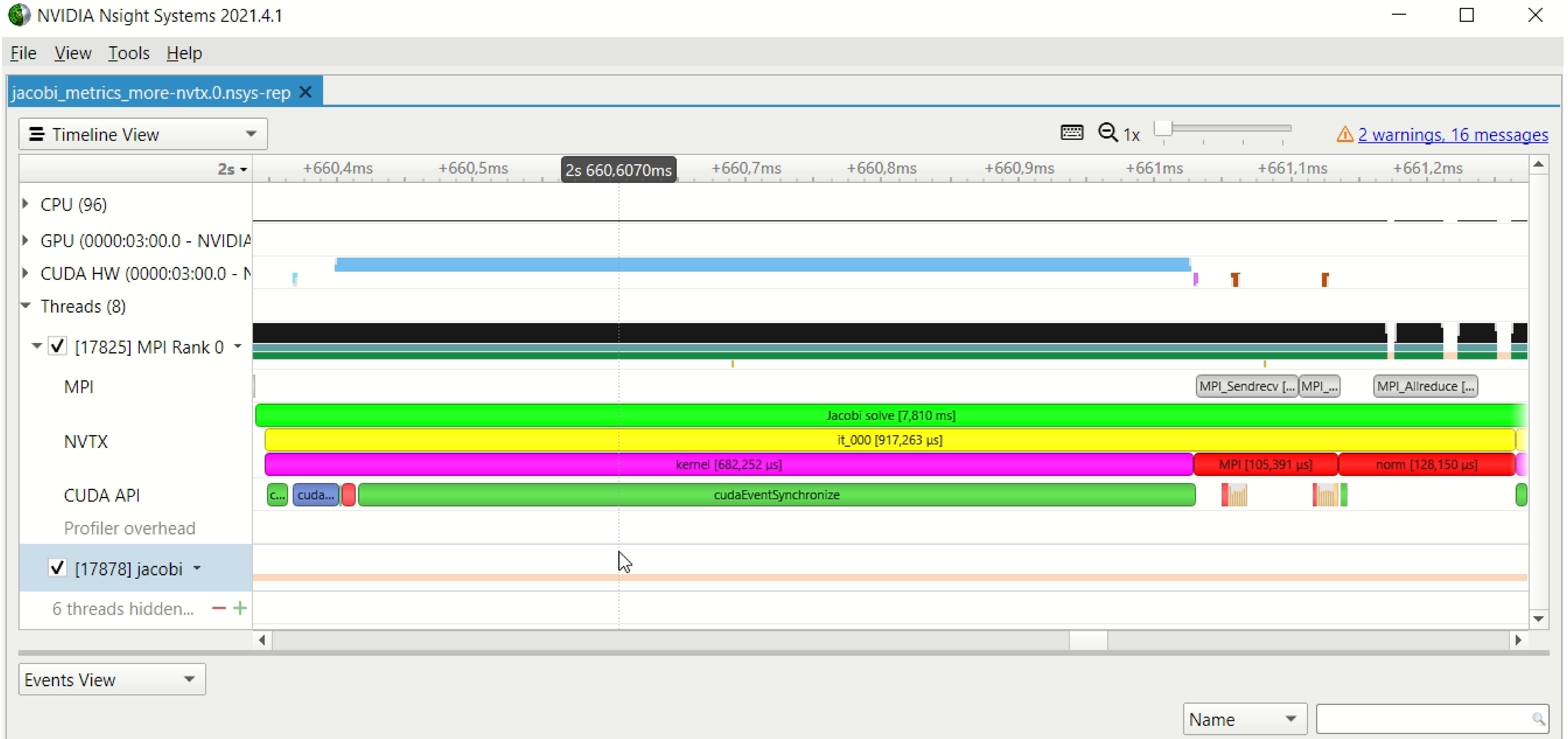
Code annotation with NVTX

- Same section of timeline as before
  - Events view: Quick navigation
- Like manual timing, only less work
- Nesting
- Correlation, filtering



Nsight Systems Workflow with NVTX

Repeating the analysis

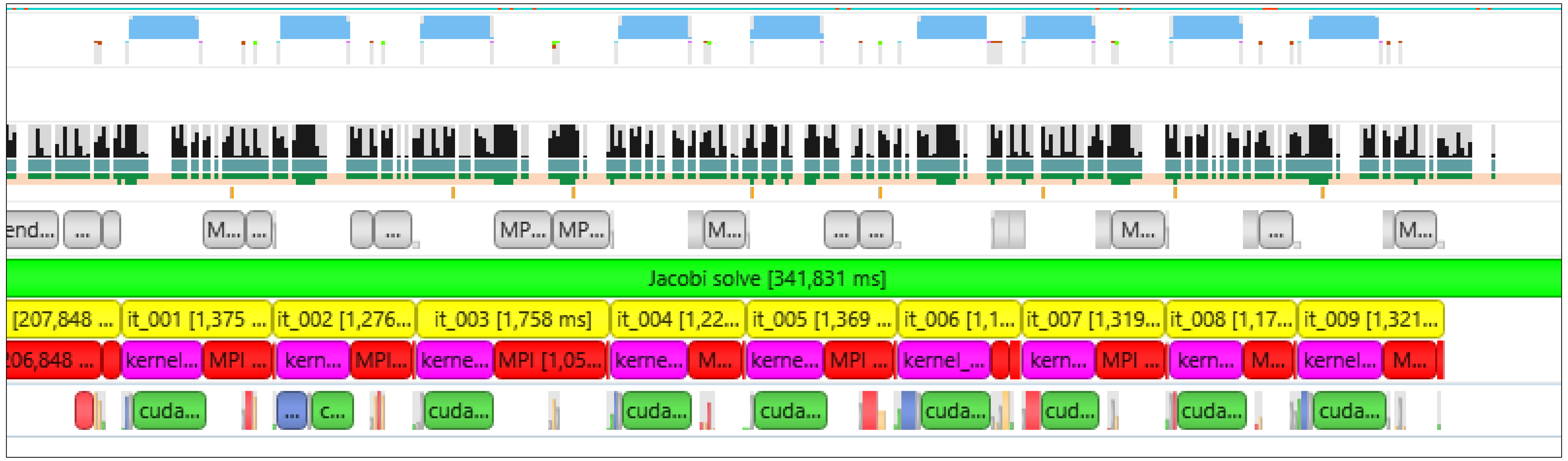


#	Name	Start	Duration	TID	Category
1	init	0,17077s	2,490 s	17825	
5	Jacobi solve	2,66034s	7,810 ms	17825	
6	it_000	2,66035s	917,263 µs	17825	
7	kernel	2,66035s	682,252 µs	17825	
8	MPI	2,66103s	105,391 µs	17825	
9	norm	2,66114s	128,150 µs	17825	
10	it_001	2,66127s	751,182 µs	17825	
14	it_002	2,66202s	767,433 µs	17825	
18	it_003	2,66279s	752,632 µs	17825	
22	it_004	2,66354s	762,572 µs	17825	

Minimizing Profile Size

Shorter time, smaller files = quicker progress

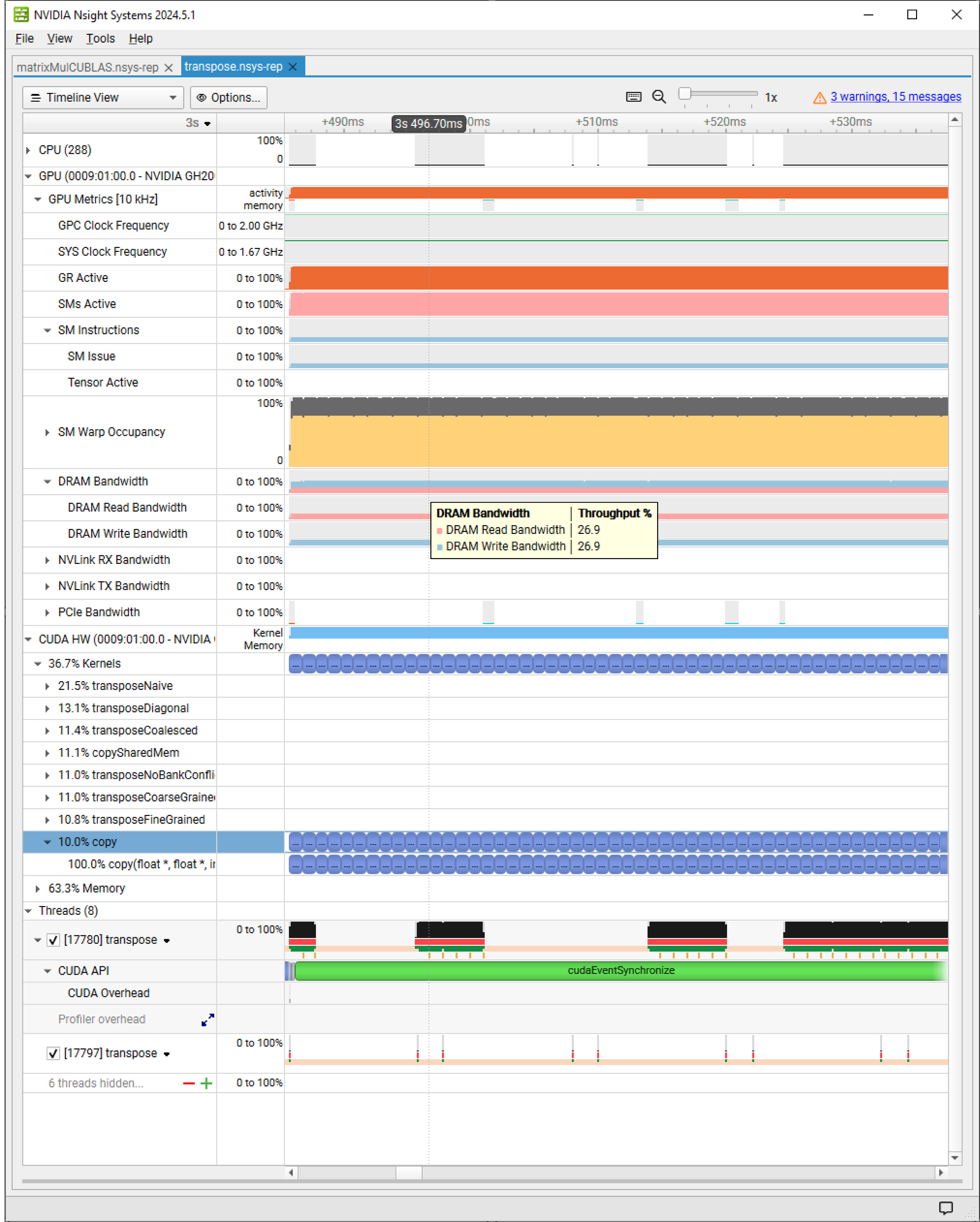
- Only profile what you need – all profilers have some overhead
  - Example: Event that occurs after long-running setup phase
- Bonus: lower number of events leads to smaller file size
- Add to nsys command line:
  - `--capture-range=nvtx --nvtx-capture=`**`any_nvtx_marker_name`**` \`
    `--env-var=NSYS_NVTX_PROFILER_REGISTER_ONLY=0 --kill none`
    - Use [NVTX registered strings](#) for best performance
- Alternatively: `cudaProfilerStart()` and `-Stop()`
  - `--capture-range=cudaProfilerApi`



GPU Metric Sampling

- Use `nsys profile … --gpu-metrics-device= …`
- With multiple GPUs per node `--gpu-metrics-device` needs to be consistent with application GPU affinity

```
export CUDA_VISIBLE_DEVICES=${SLURM_LOCALID}
nsys profile ... --gpu-metrics-device=${SLURM_LOCALID} ./a.out
```

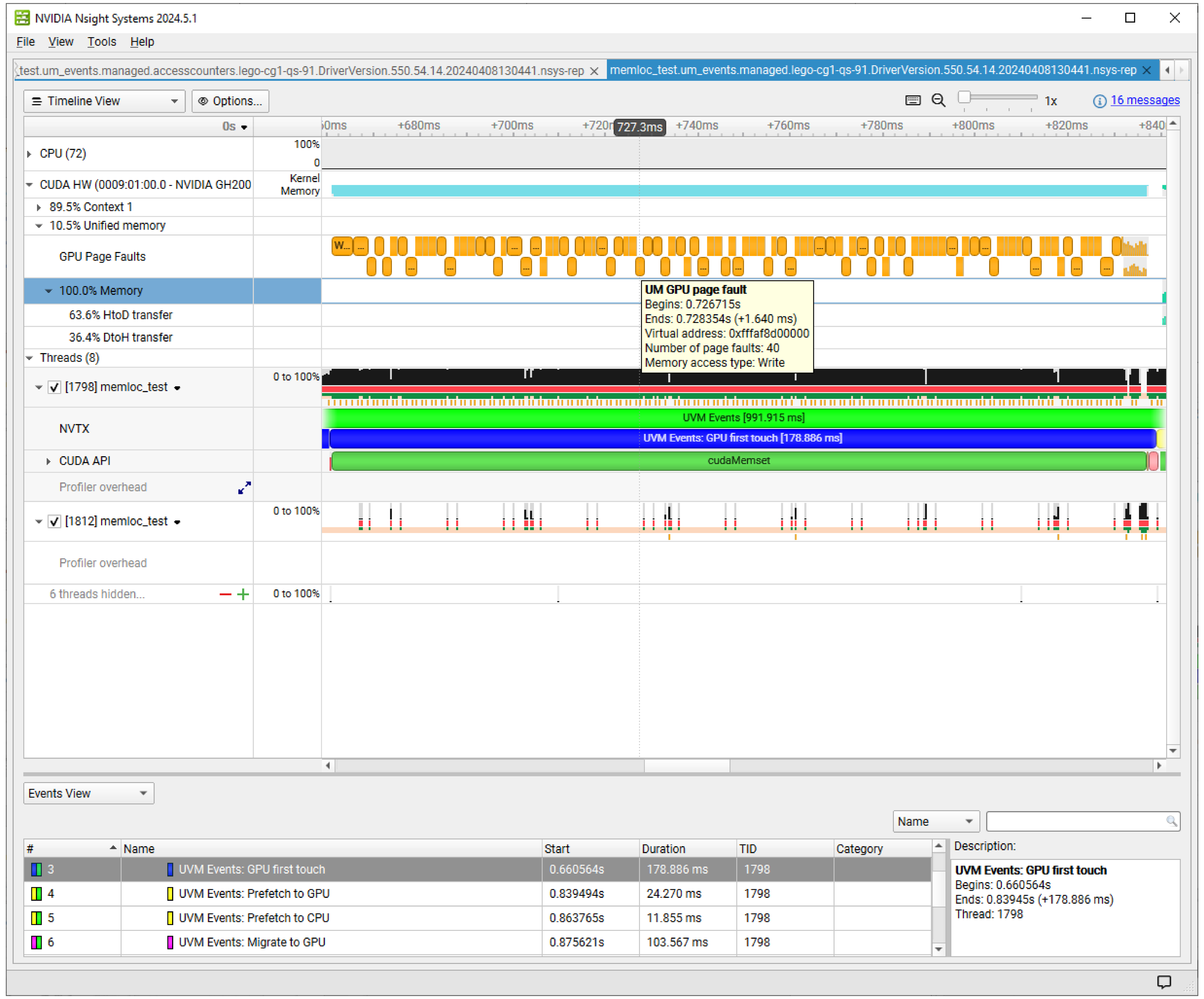


Unified Memory Profiling

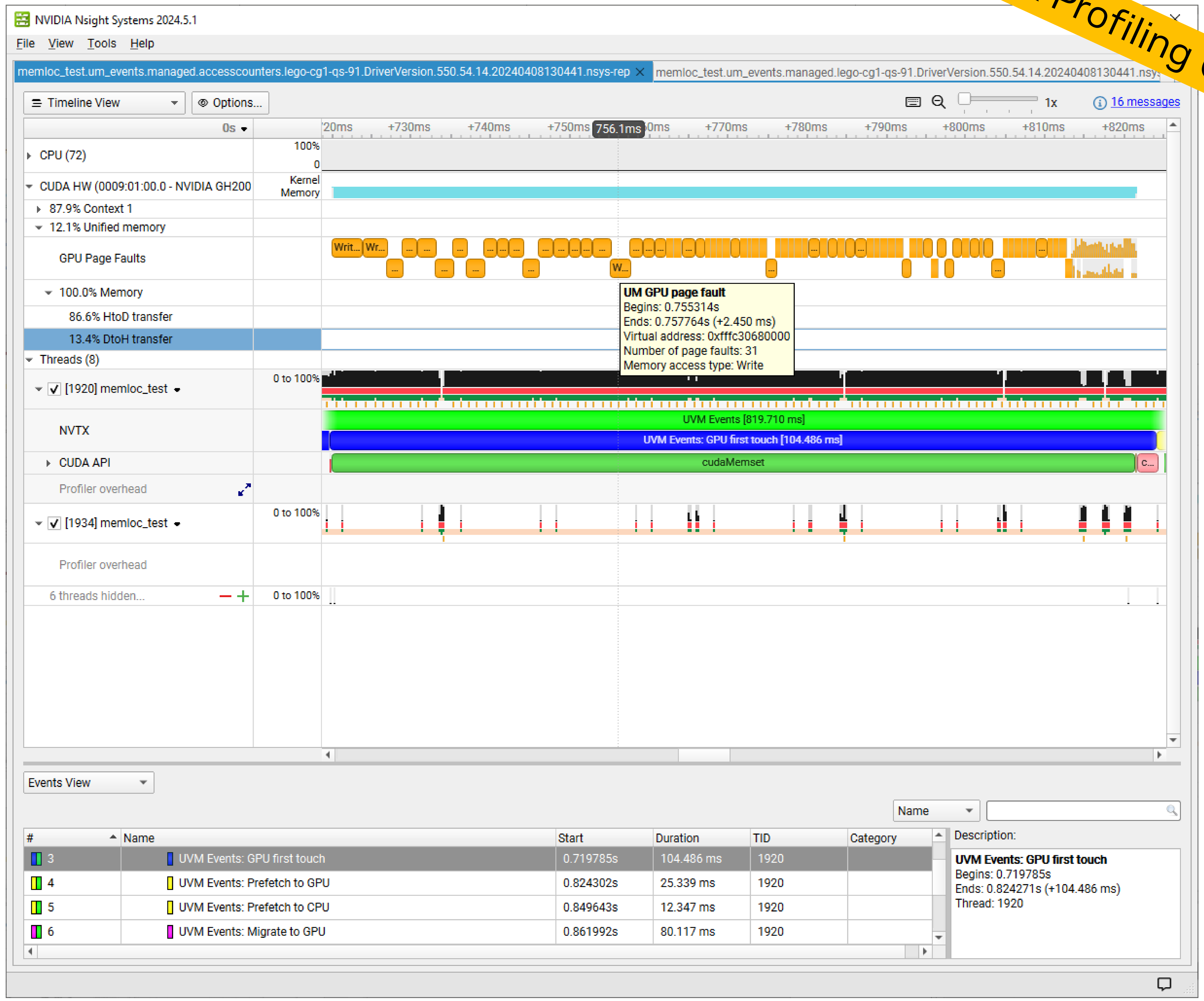
`nsys profile … --cuda-um-cpu-page-faults true --cuda-um-gpu-page-faults true`

Check Profiling Overhead

Fault Based Migrations



Access Counter Based Migration



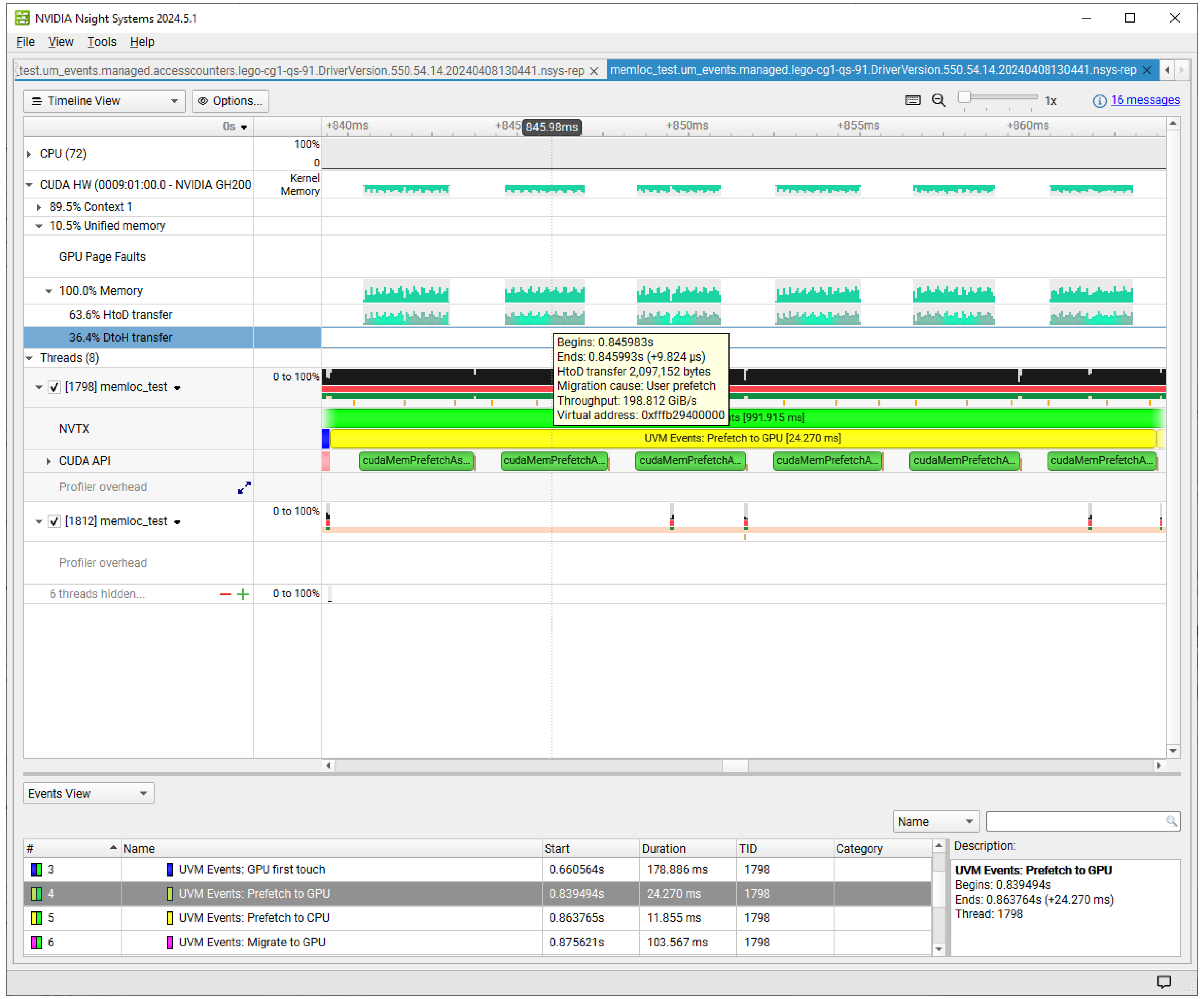
NVIDIA

Unified Memory Profiling

`nsys profile … --cuda-um-cpu-page-faults true --cuda-um-gpu-page-faults true`

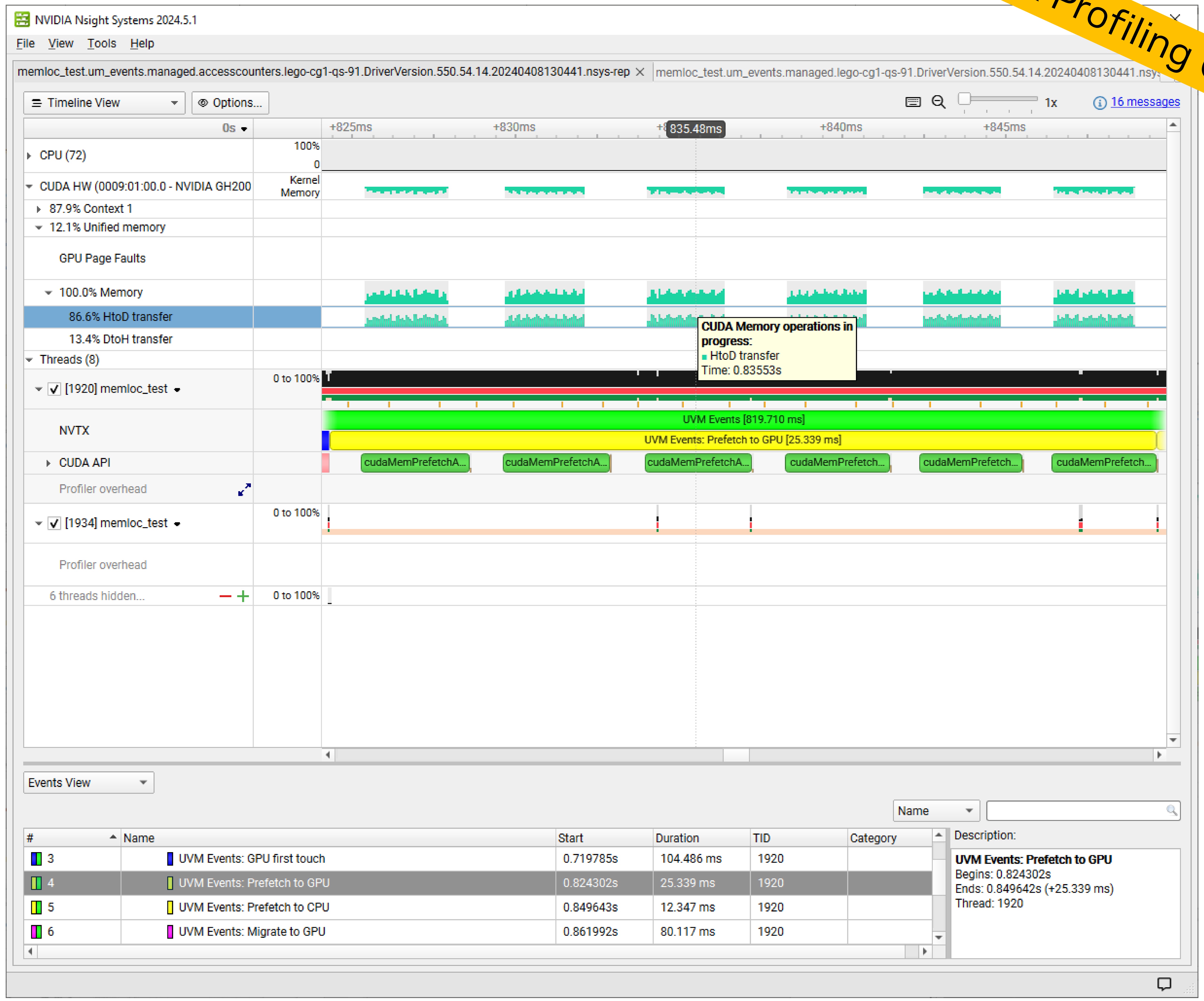
Check Profiling Overhead

Fault Based Migrations



#	Name	Start	Duration	TID	Category
3	UVM Events: GPU first touch	0.660564s	178.886 ms	1798	
4	UVM Events: Prefetch to GPU	0.839494s	24.270 ms	1798	
5	UVM Events: Prefetch to CPU	0.863765s	11.855 ms	1798	
6	UVM Events: Migrate to GPU	0.875621s	103.567 ms	1798	

Access Counter Based Migration

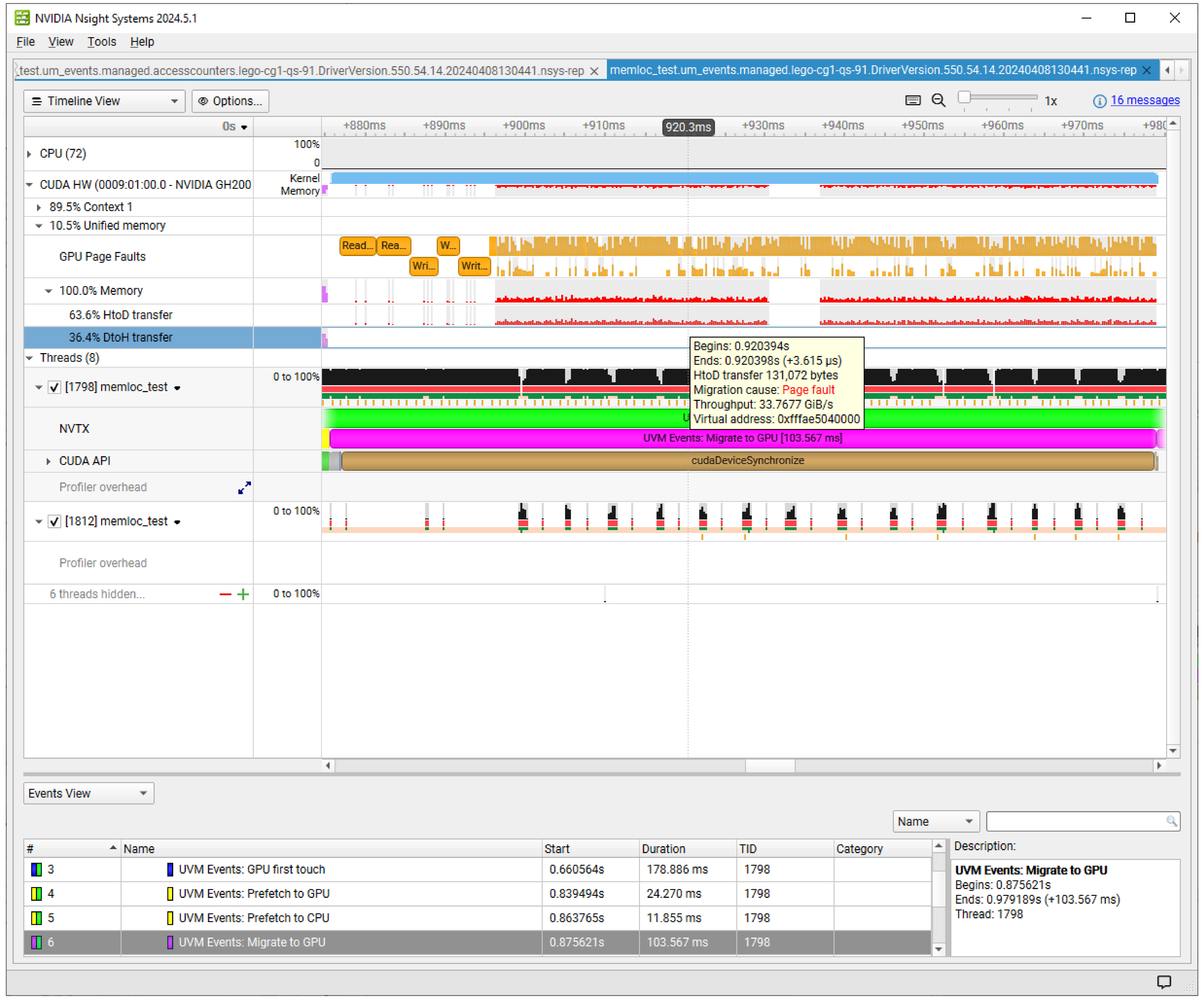


#	Name	Start	Duration	TID	Category
3	UVM Events: GPU first touch	0.719785s	104.486 ms	1920	
4	UVM Events: Prefetch to GPU	0.824302s	25.339 ms	1920	
5	UVM Events: Prefetch to CPU	0.849643s	12.347 ms	1920	
6	UVM Events: Migrate to GPU	0.861992s	80.117 ms	1920	

Unified Memory Profiling

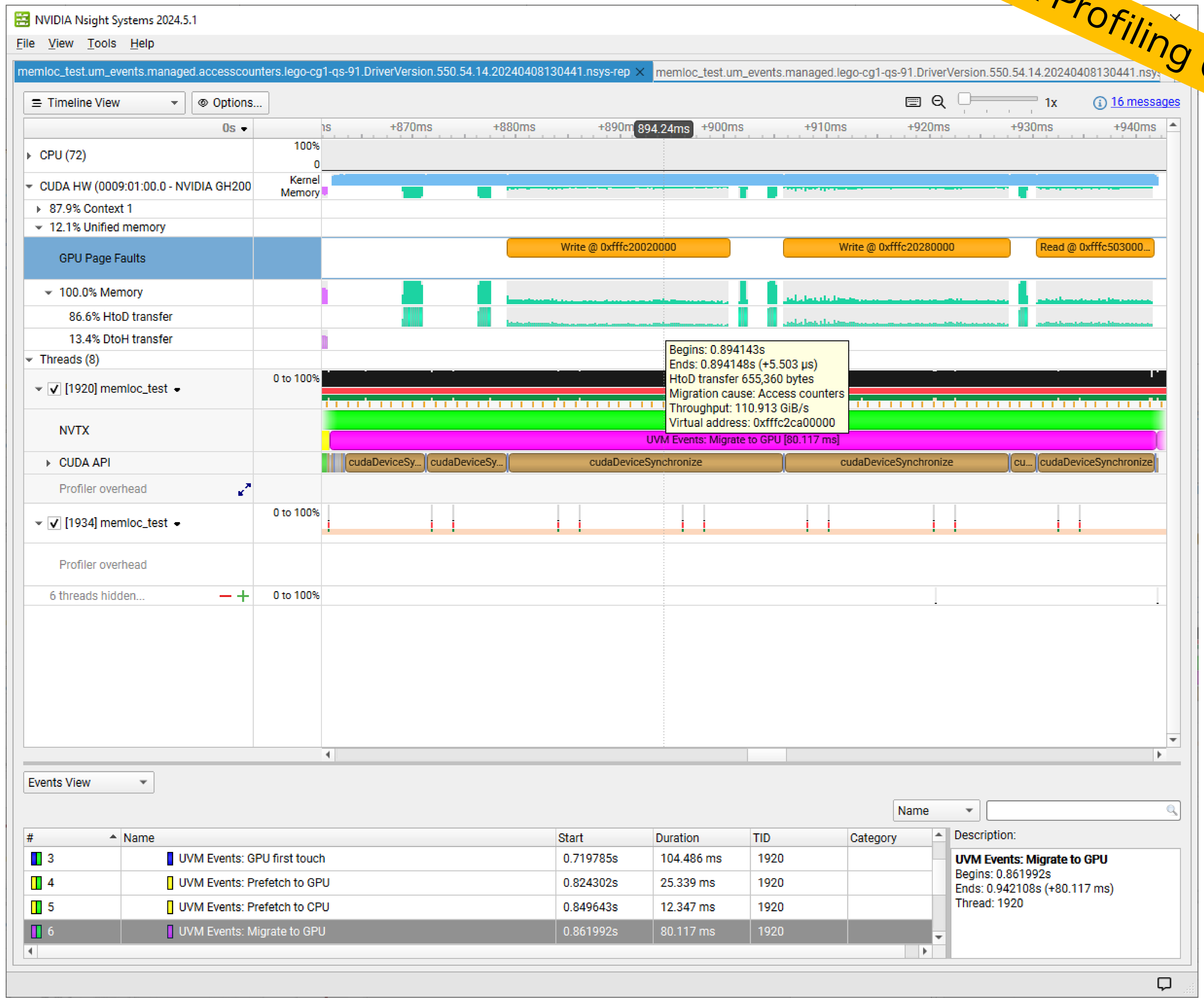
`nsys profile … --cuda-um-cpu-page-faults true --cuda-um-gpu-page-faults true`

Fault Based Migrations



Access Counter Based Migration

Check Profiling Overhead



Approaches for multi-process tools

- Tools usually run on a single process – adapt for highly distributed applications?
  - Bugs in parallel programs are often serial bugs in disguise
- Common MPI paradigm: Workload distributed; bug classes/performance similar for all processes
  - Not: Load imbalance, parallel race conditions; require parallel tools
- Ergo: Run tool N times in parallel, have N output files, only look at 1 (or 2, …)
  - `%q{ENV_VAR}` supported by all the NVIDIA tools discussed here, embed environment variable in file name
  - `ENV_VAR` should be one set by the process launcher, unique ID
  - Evaluated only once tool starts running (on compute node) – not when launching job
- Other tools: Use a launcher script, for late evaluation

```
srun nsys profile --trace=cuda,nvtx,mpi --force-overwrite=true \
        --output=my_report.%q{SLURM_PROCID}                    \
        ./jacobi -niter 10
```

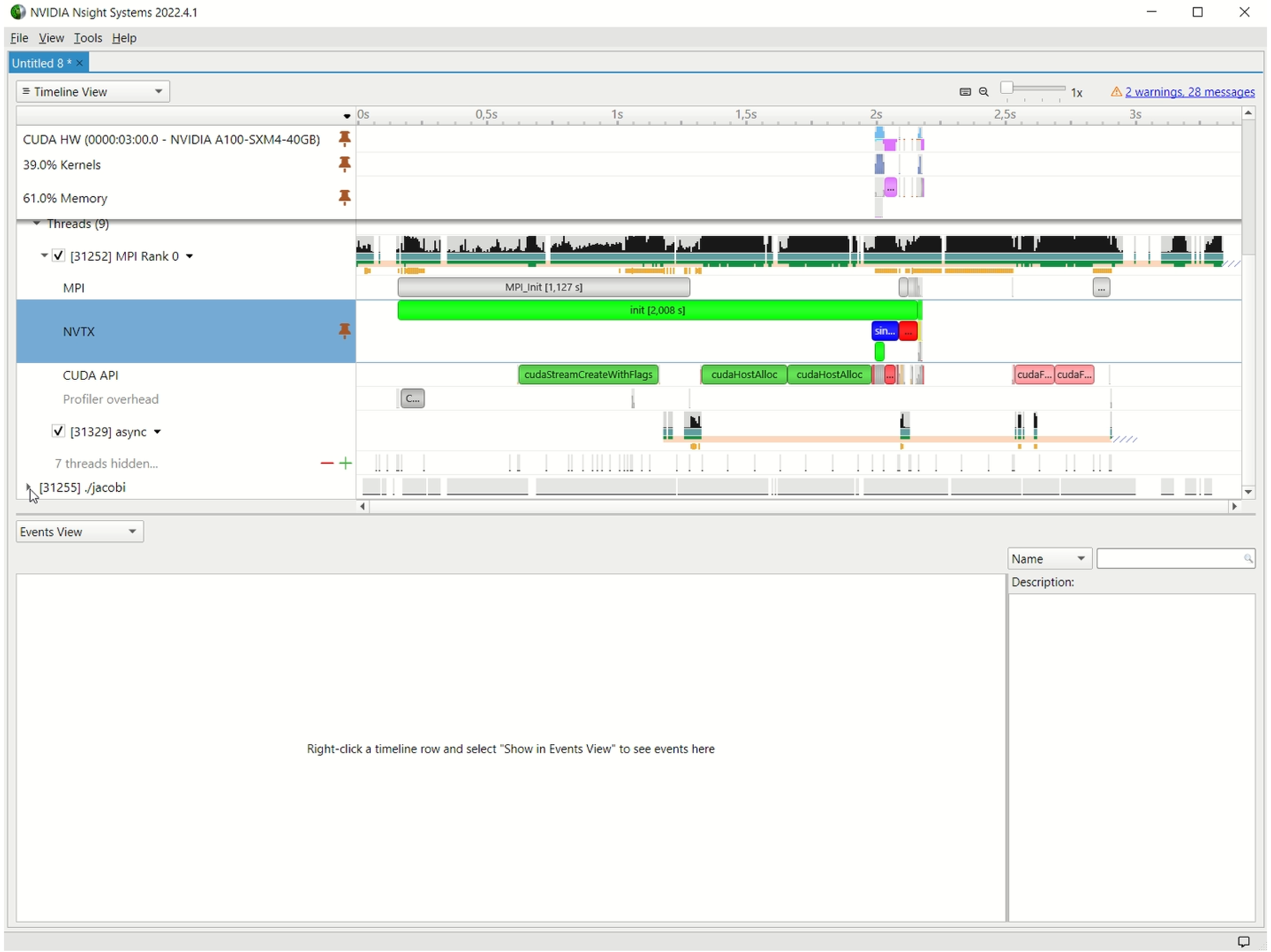
```
OpenMPI:
OMPI_COMM_WORLD_RANK
OMPI_COMM_WORLD_LOCAL_RANK

MVAPICH2:
MV2_COMM_WORLD_RANK
MV2_COMM_WORLD_LOCAL_RANK

Slurm:
SLURM_PROCID
SLURM_LOCALID
```

https://www.open-mpi.org/faq/?category=running#mpi-environmental-variables
http://mvapich.cse.ohio-state.edu/static/media/mvapich/mvapich2-2.2-userguide.html#x1-32100013
https://slurm.schedmd.com/srun.html#SECTION_OUTPUT-ENVIRONMENT-VARIABLES

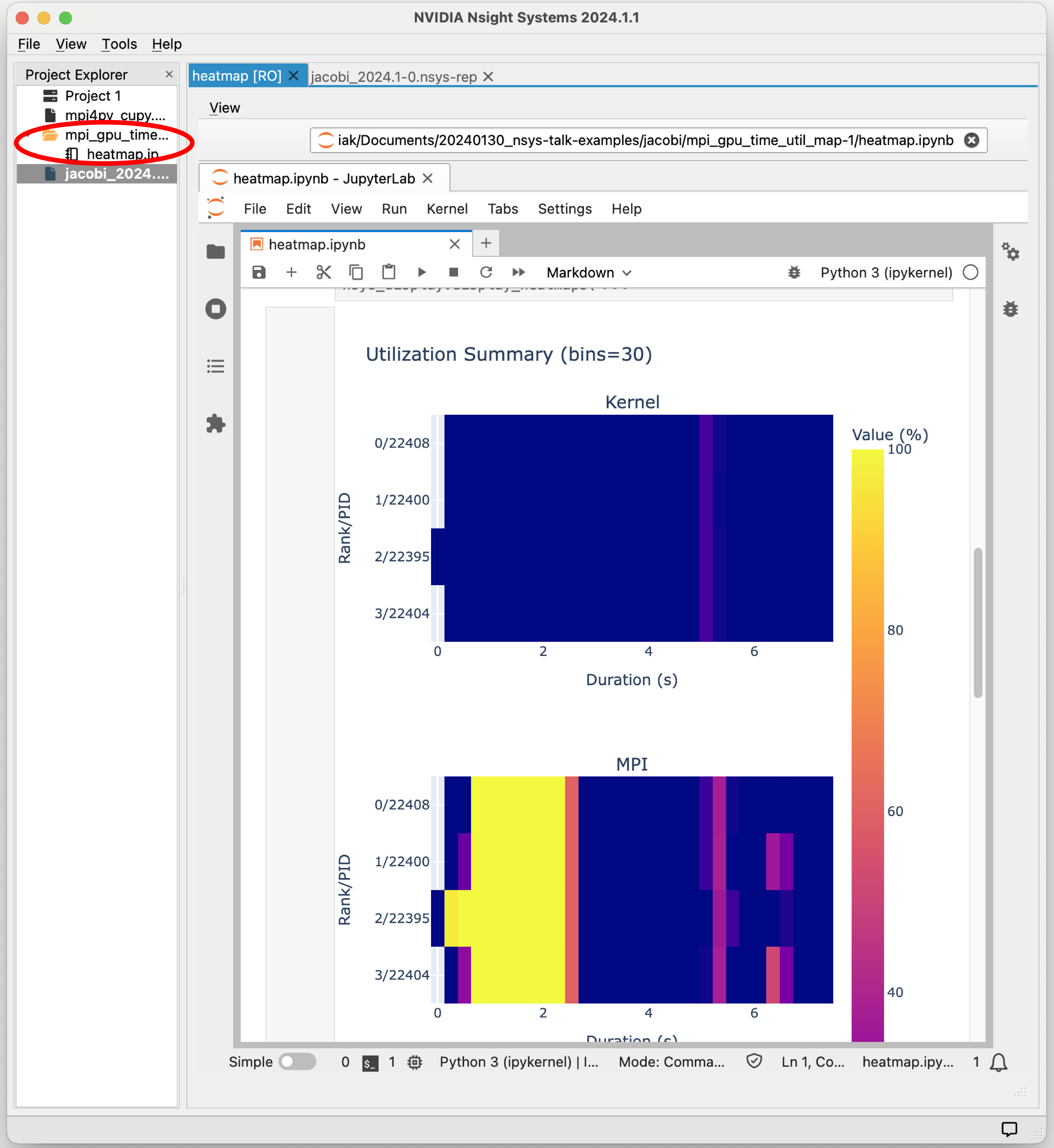
Using Multiple Reports in Nsight Systems



Multi-report analysis recipes

Preview Feature

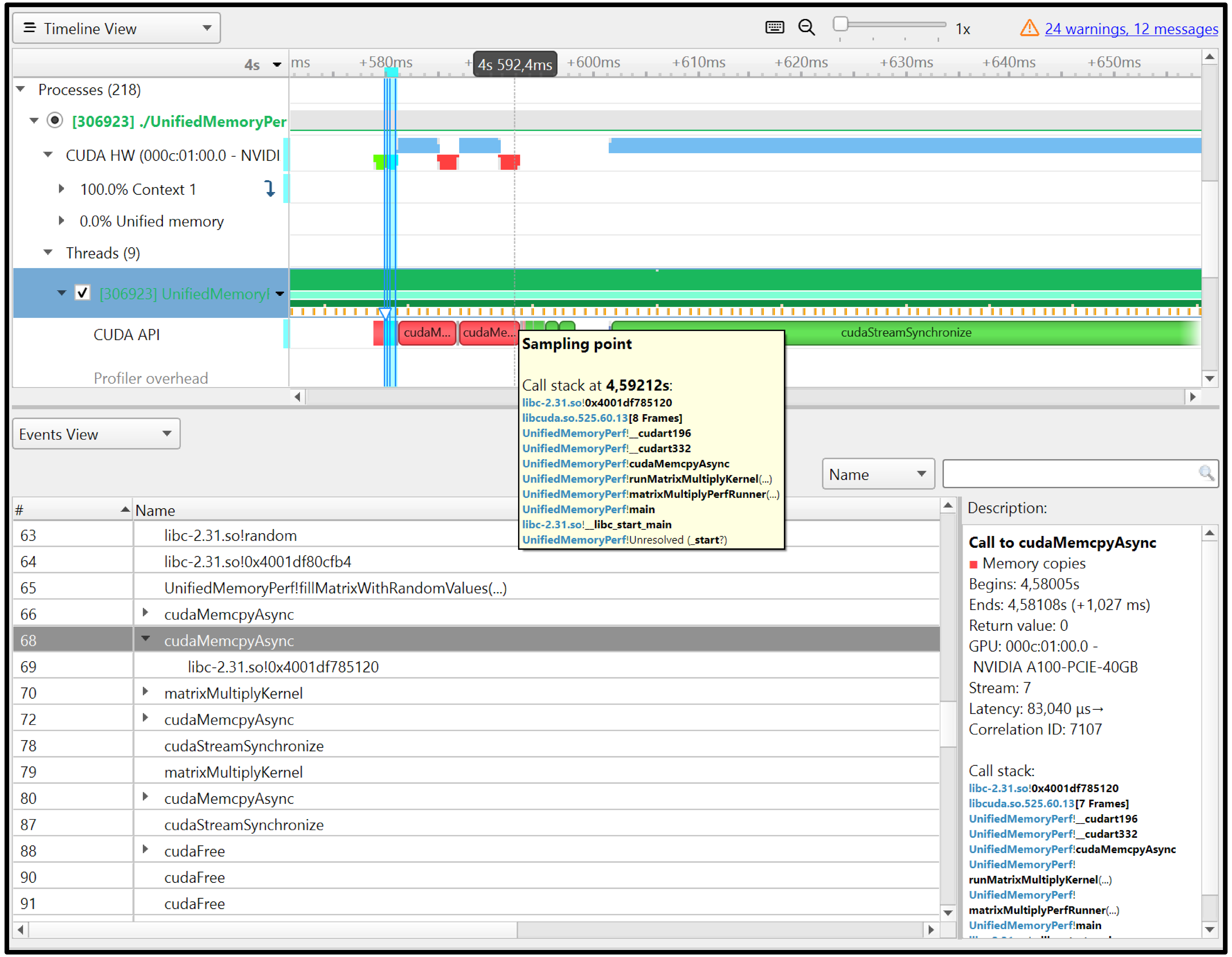
- Installation guide: https://docs.nvidia.com/nsight-systems/InstallationGuide/index.html#installing-multi-report-analysis-system
- Bundled installer script – install.py, put dependencies into Python venv
  - Can be done by users themselves – but could also be provisioned
- Run one of the recipes (or customize), example:
  - nsys recipe mpi_gpu_time_util_map --input *.nsys-rep
- Load generated .ipynb (embedded in nsys GUI)



CPU Profiling

Analyzing host-side call trees

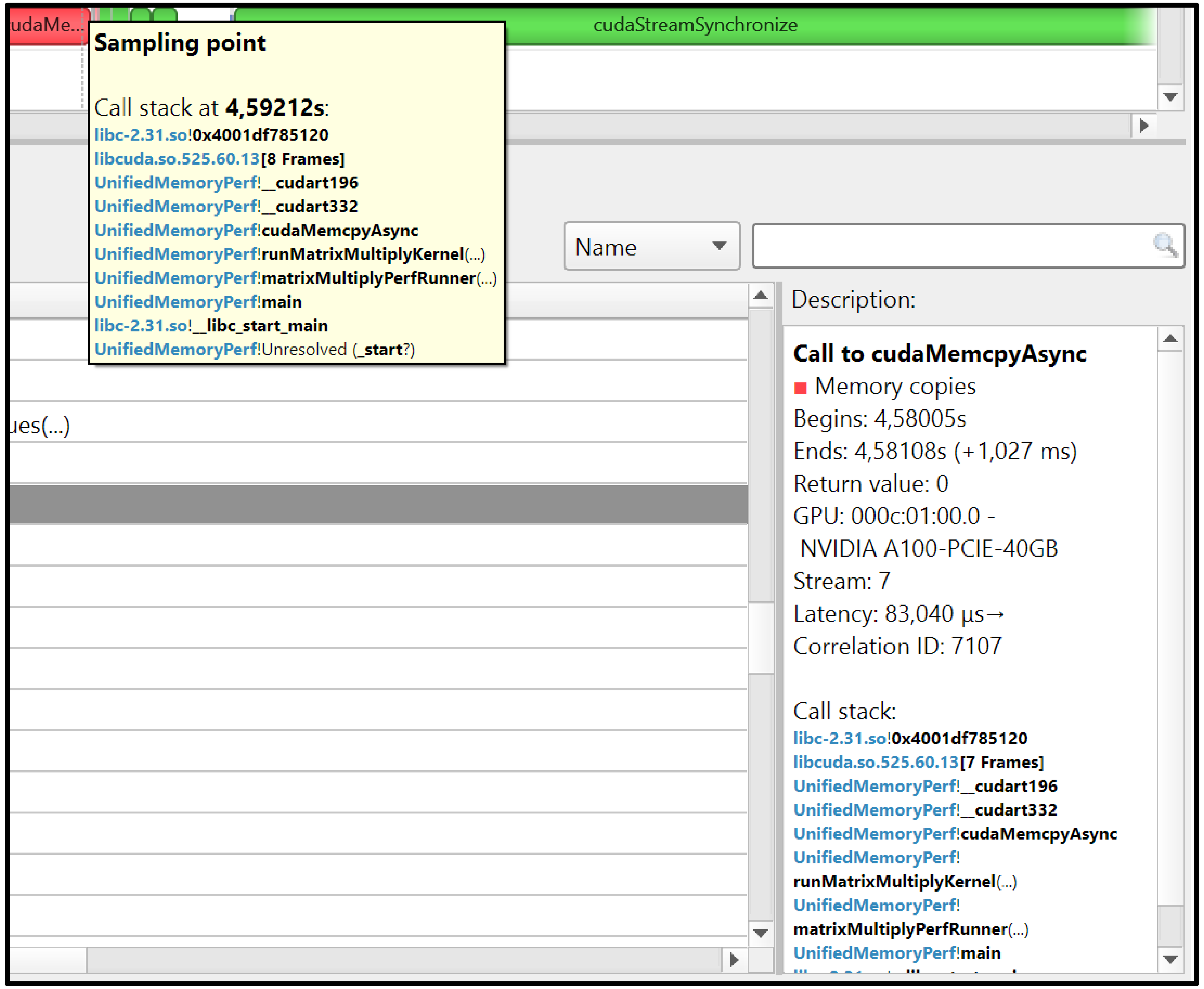
- Nsight Systems by default collects stack samples
  - --sample=none to disable
- Display process/thread in events view or use tooltips



CPU Profiling

Zooming in

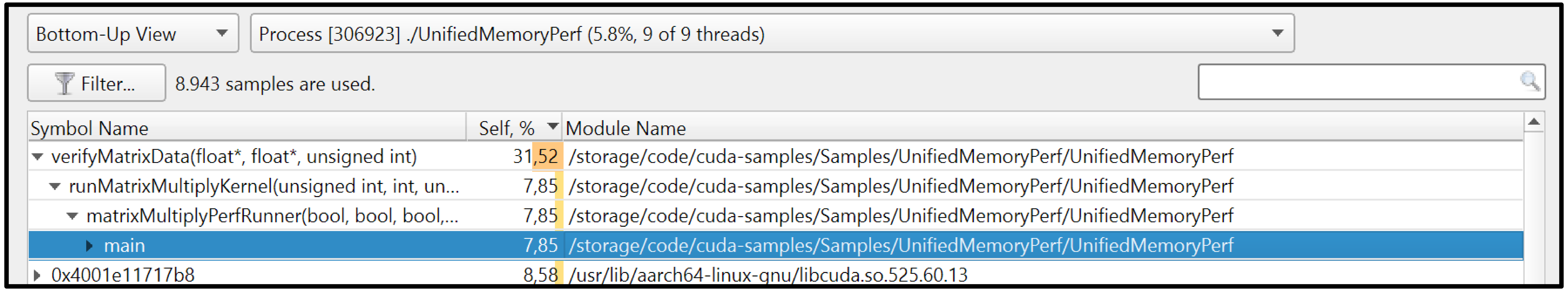
- Example: Backtrace in CUDA sample
- Call stack leading up to cudaMemcpyAsync
- Periodic sampling: low overhead, but may miss some call chains
  - --cudabacktrace ensures CUDA API calls show backtrace, potentially higher overhead

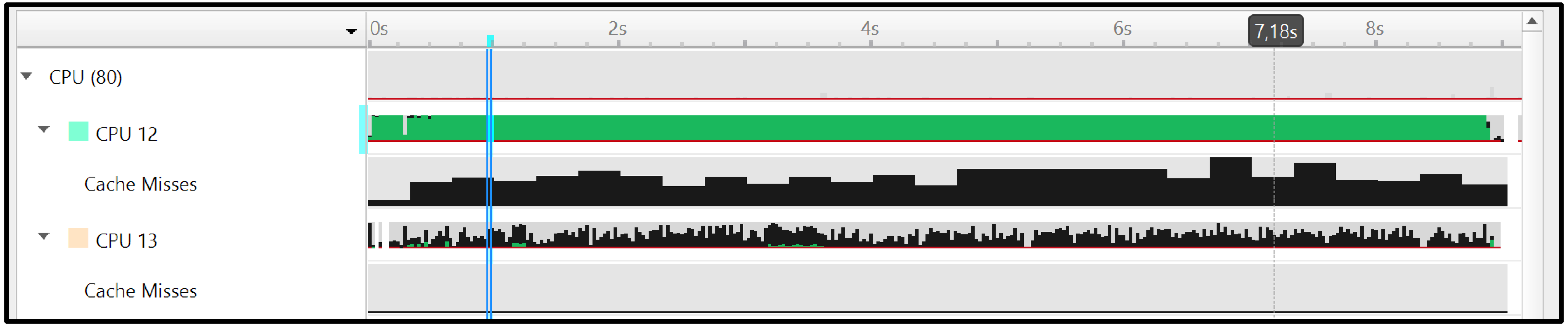


CPU Profiling

Analyzing host-side call trees

- CPU bottom-up (or top-down) profiling based on samples
- Linux perf events, e.g. cache misses
  - https://docs.nvidia.com/nsight-systems/UserGuide/index.html#cpu-linuxperf





CPU Core Metrics

Profiling on Grace

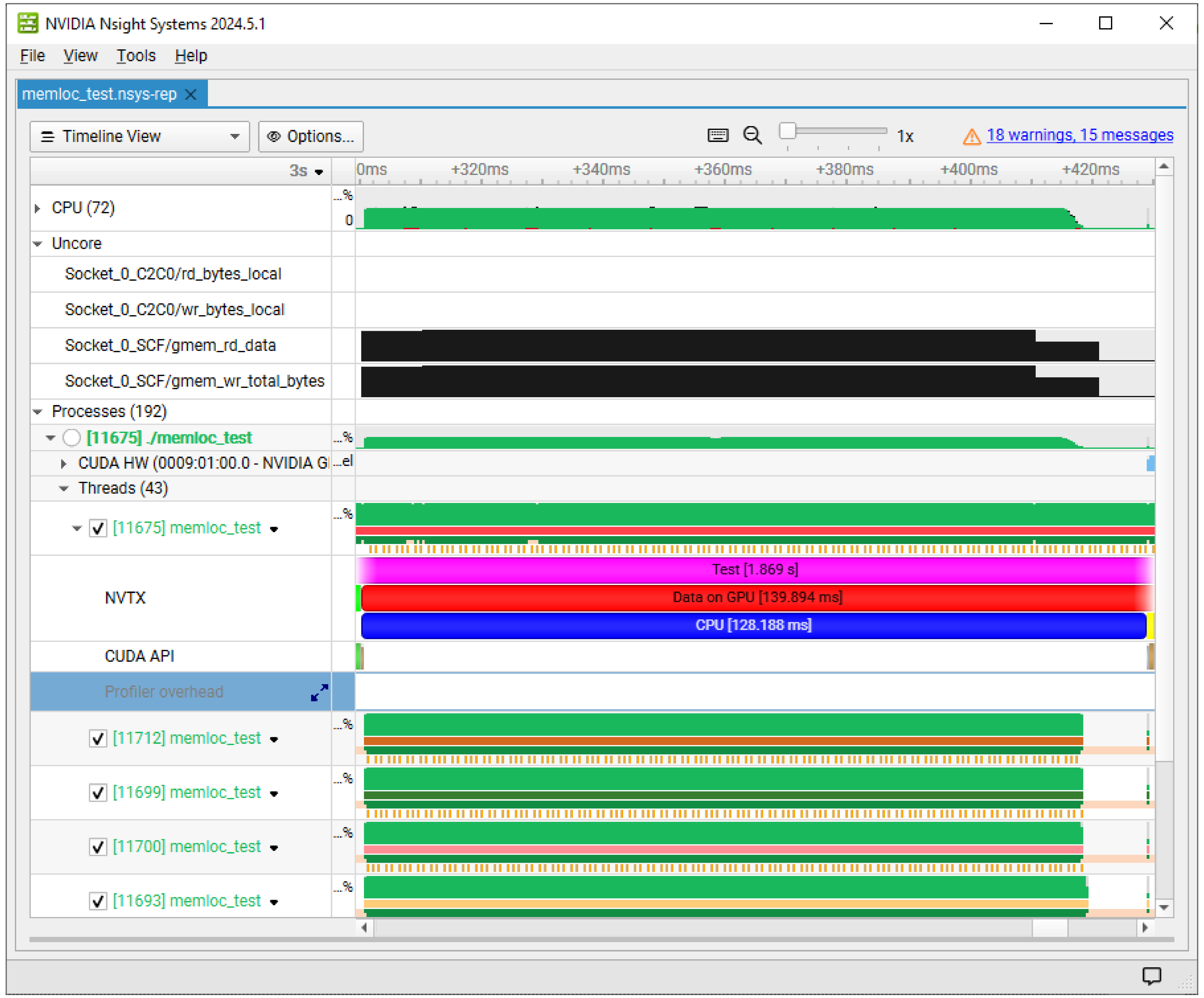
- `nsys profile --cpu-core-metrics=help (--cpu-core-events=help) --cpu-socket-metrics=help (--cpu-socket-events=help)`
  - Lists available metrics (events) on the CLI
- Generally recommended read: https://docs.nvidia.com/grace-perf-tuning-guide/index.html
  - In particular, see https://docs.nvidia.com/grace-perf-tuning-guide/index.html#measuring-workload-performance-with-hardware-performance-counters

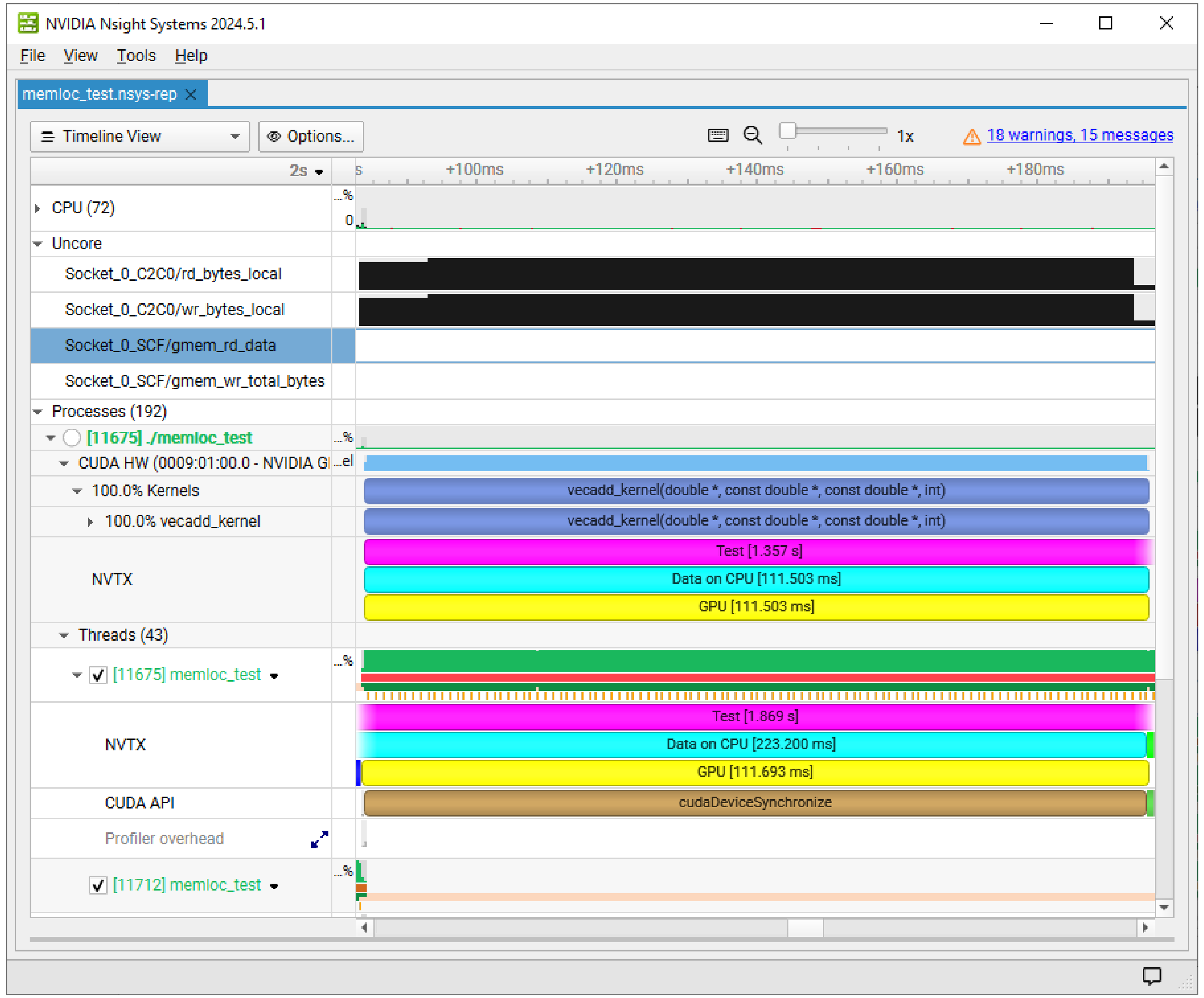
Measuring Workload Performance with Hardware Performance Counters

Many software performance analysis tools rely on event counts from hardware performance monitoring units (PMUs) to characterize workload performance. This chapter provides information about how data from PMUs can be gathered and combined to form metrics for performance optimization. For simplicity, the Linux perf tool is used, but the same metrics can be used in any tool that gathers hardware performance events from PMUs, for example, NVIDIA NSIGHT Systems.

CPU Socket Metrics

--cpu-socket-events=61,71,265,273





NVIDIA