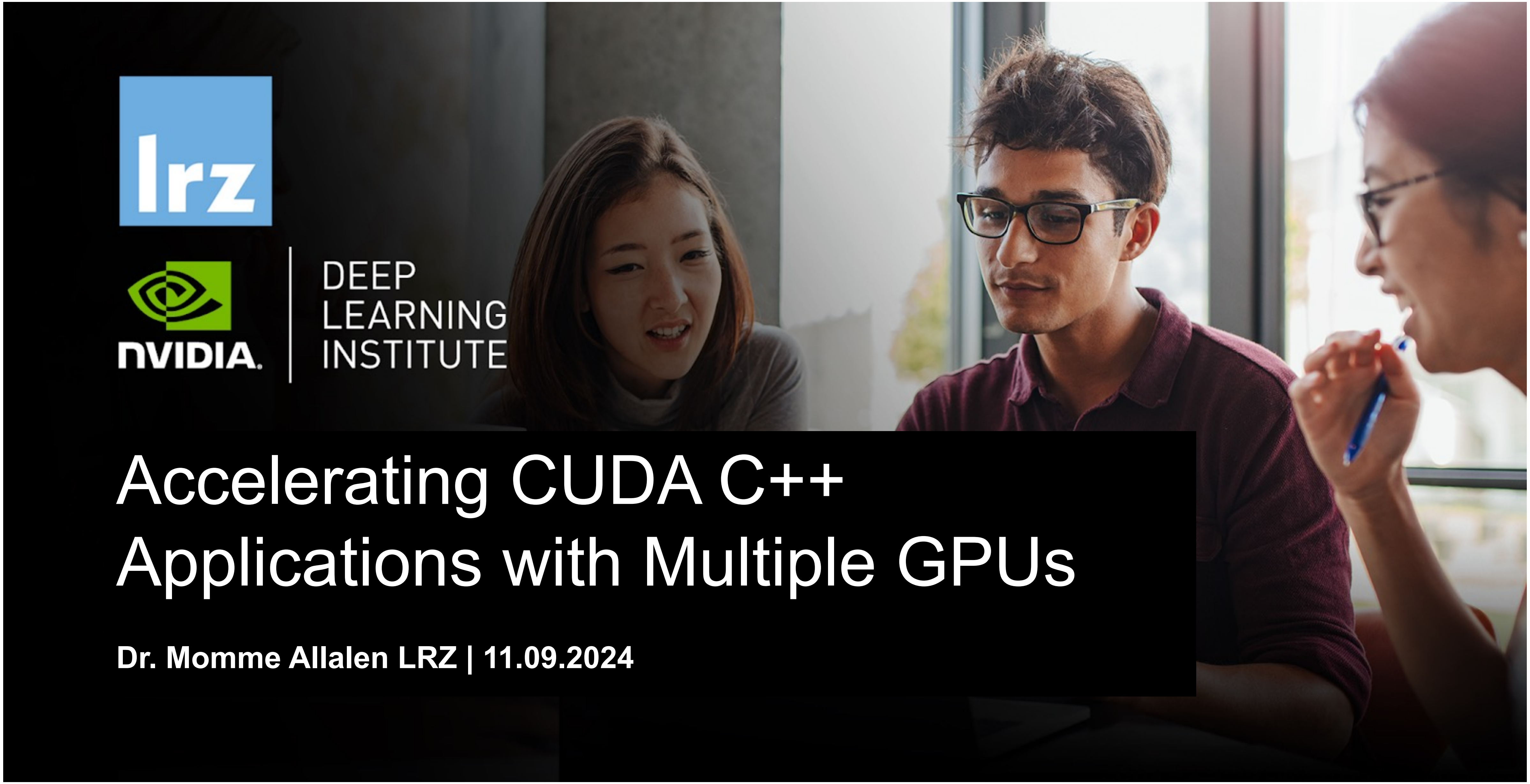
lrz

Leibniz-Rechenzentrum
der Bayerischen Akademie der Wissenschaften

NVIDIA.

DEEP
LEARNING
INSTITUTE



lrz

NVIDIA | DEEP LEARNING INSTITUTE

Accelerating CUDA C++ Applications with Multiple GPUs

Dr. Momme Allalen LRZ | 11.09.2024

Accelerating CUDA C++ Applications with Multiple GPUs

- Learn how to accelerate your **CUDA C++ application** by using multiple GPUs.
- Discover how to combine concurrent computation and memory transfers, computation can be scaled across multiple GPUs without increasing the cost of memory transfers.
- Maximize performance on multi-GPU systems, whether using cloud based servers or on **NVIDIA DGX** systems, these techniques enable you to achieve peak performance from GPU-accelerated applications.
- Implement these single-node, multi-GPU techniques before scaling your applications across multiple nodes.
- Write efficient and correct **CUDA C++** applications. This lecture covers how to write **CUDA C++** applications that efficiently and correctly utilise all available GPUs in a single node, dramatically improving the performance of your applications and making the most cost-effective use of systems with multiple GPUs.

Workshop Webpage

- **Lecture material will be made available under:**
  - https://tinyurl.com/hdli2s24
- **Access CUDA C/C++ Code :**
  - See the **Chat Window**

THE NSIGHT SUITE COMPONENTS

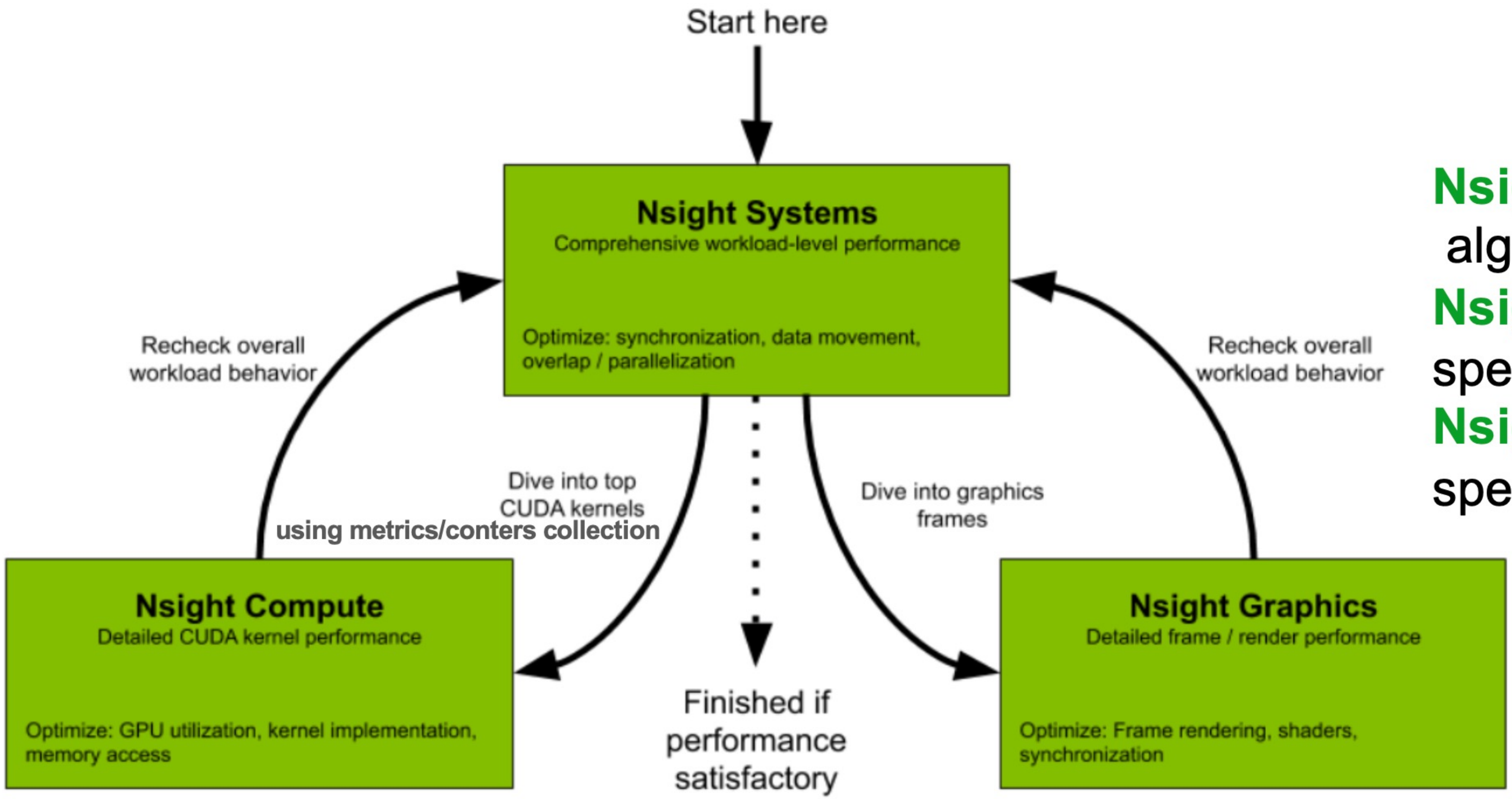

Start here
Nsight Systems
Comprehensive workload-level performance
Optimize: synchronization, data movement, overlap / parallelization
Recheck overall workload behavior
Recheck overall workload behavior
Dive into top CUDA kernels
using metrics/conters collection
Dive into graphics frames
Nsight Compute
Detailed CUDA kernel performance
Optimize: GPU utilization, kernel implementation, memory access
Finished if performance satisfactory
Nsight Graphics
Detailed frame / render performance
Optimize: Frame rendering, shaders, synchronization

Figure 1. Flowchart describing working with new NVIDIA Nsight tools for performance optimization

Nsight Systems – Analyze application algorithm system wide

Nsight Compute – Debug/&optomize specific CUDA kernels

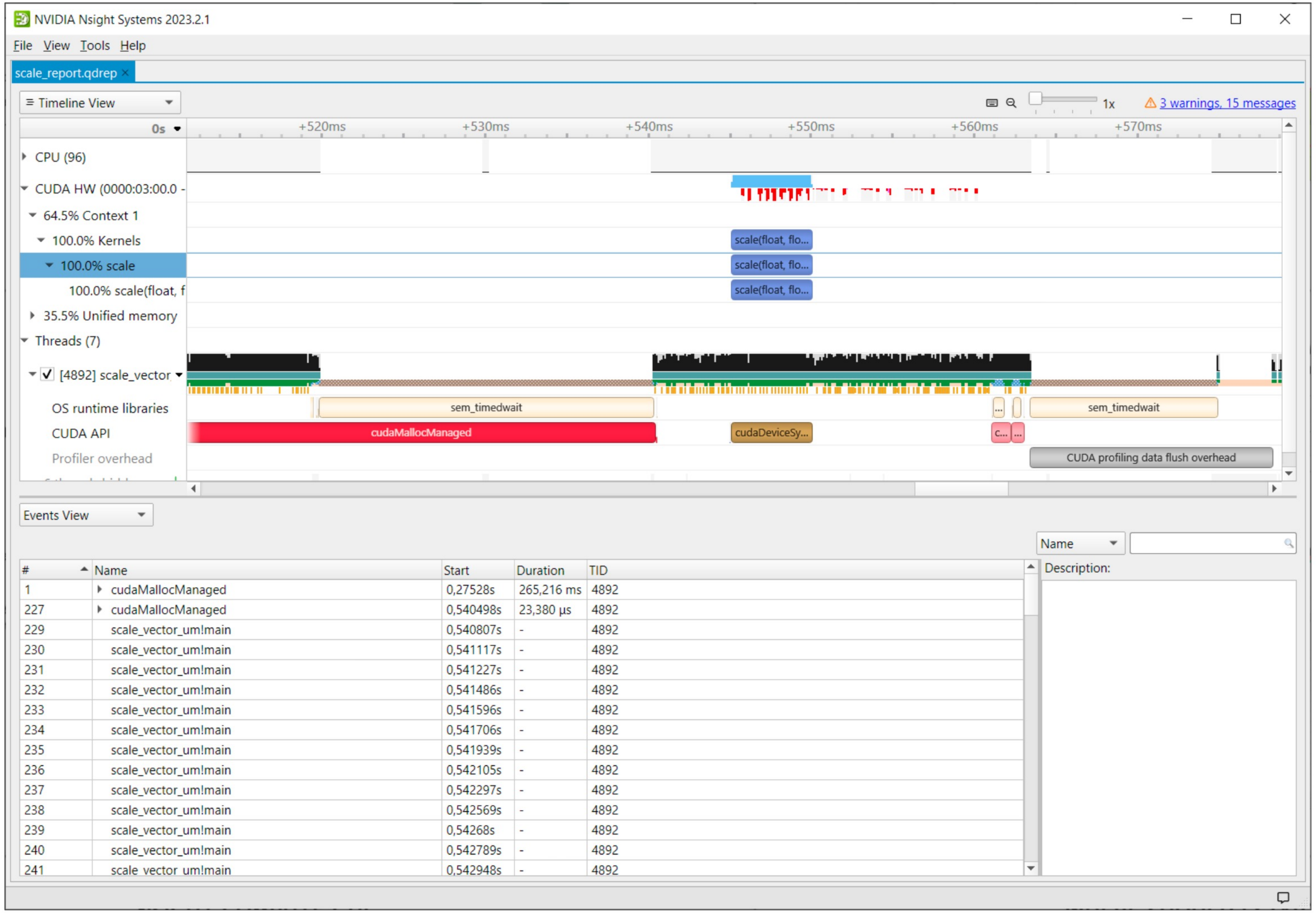
Nsight Graphics – Debug/&optimize specific graphics workloads

nvprof replaced with **nsys –profile....**

https://developer.nvidia.com/nsight-systems

Nsight Systems GUI

Main timeline view, Events View



#	Name	Start	Duration	TID
1	cudaMallocManaged	0,27528s	265,216 ms	4892
227	cudaMallocManaged	0,540498s	23,380 µs	4892
229	scale_vector_um!main	0,540807s	-	4892
230	scale_vector_um!main	0,541117s	-	4892
231	scale_vector_um!main	0,541227s	-	4892
232	scale_vector_um!main	0,541486s	-	4892
233	scale_vector_um!main	0,541596s	-	4892
234	scale_vector_um!main	0,541706s	-	4892
235	scale_vector_um!main	0,541939s	-	4892
236	scale_vector_um!main	0,542105s	-	4892
237	scale_vector_um!main	0,542297s	-	4892
238	scale_vector_um!main	0,542569s	-	4892
239	scale_vector_um!main	0,54268s	-	4892
240	scale_vector_um!main	0,542789s	-	4892
241	scale_vector_um!main	0,542948s	-	4892

- Nsight Systems is an extremely low overhead profiling Tool across any number of CPUs and GPUs.
- Nsight System is your first stop on your profiling workflow, inspect your algorithm timing and GPU interaction and identify a large number of opportunities for optimization.

NSIGHT SYSTEMS

- Provides users with a more complete view of how their codes balance workload across multiple CPUs and GPUs

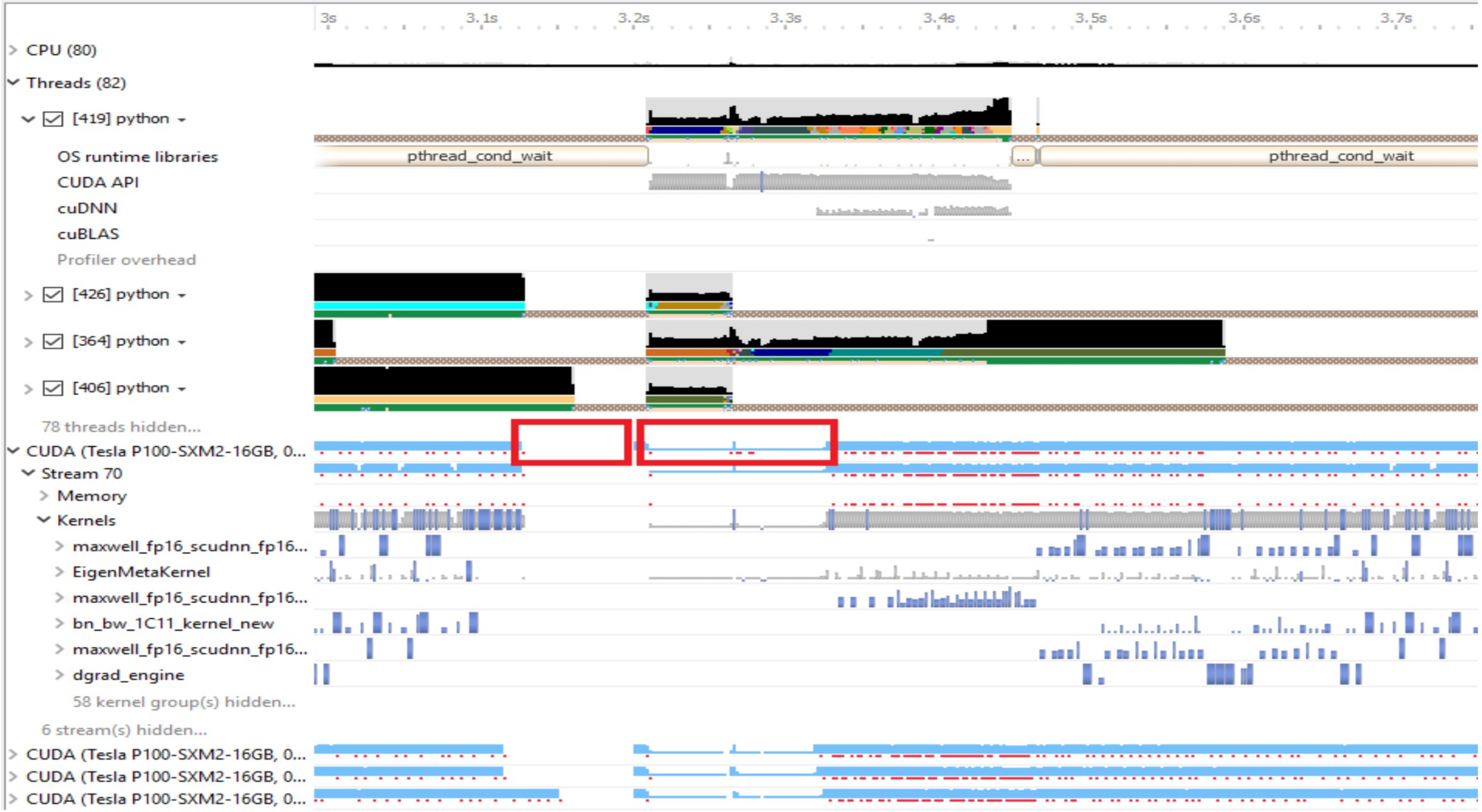
- Locate optimization opportunities, helps and allows to identify issues such as:
  - *GPU starvation*
  - *Insufficient CPU parallelisation or pipelining*
  - *Unexpectedly expensive CPU or GPU algorithm*
  - *Unnecessary GPU synchronization*

- The tool uses low overhead tracing and sampling techniques to collect process and thread activity and visualize millions of events on a very fast GUI timeline

- Correlates that data across CPU cores and GPU streams, allowing users to investigate bottlenecks.
- Multi-platform: Linux & Windows, x86-64, Tegra, Power, MacOSX (host only)

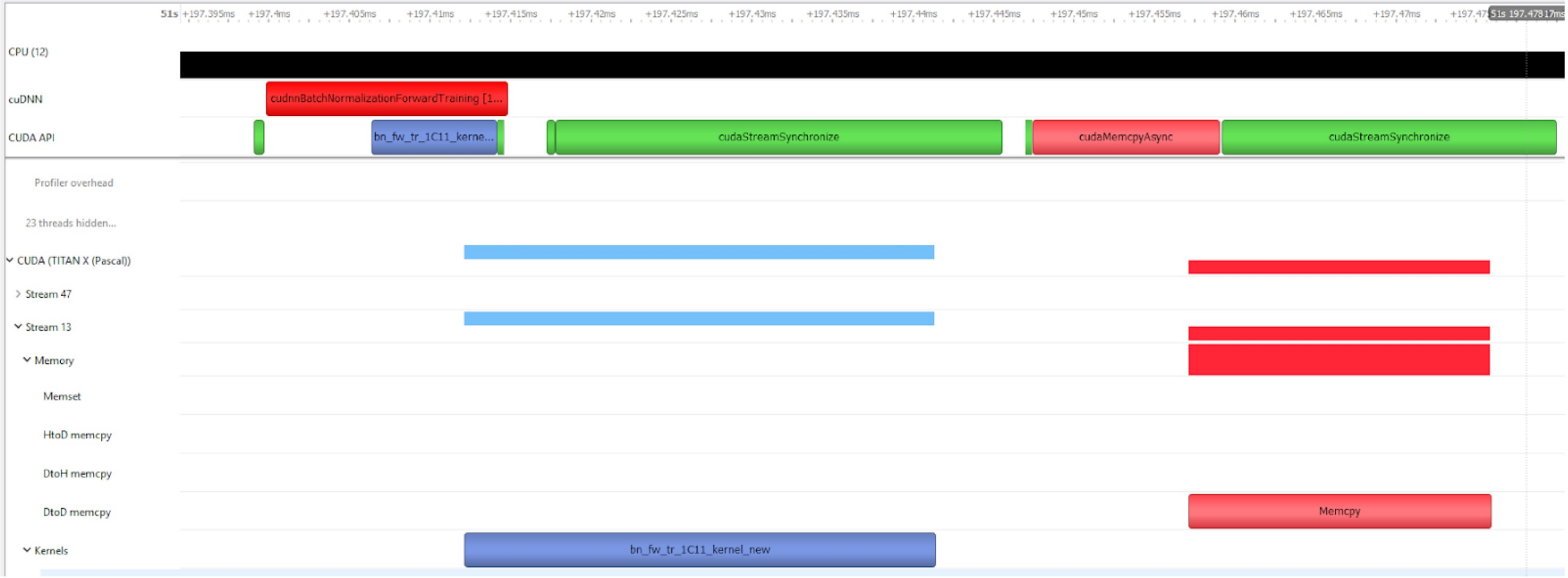
https://developer.nvidia.com/nsight-systems

GPU Starvation Investigations



https://developer.nvidia.com/nsight-systems

Unnecessary GPU Synchronisation Calls



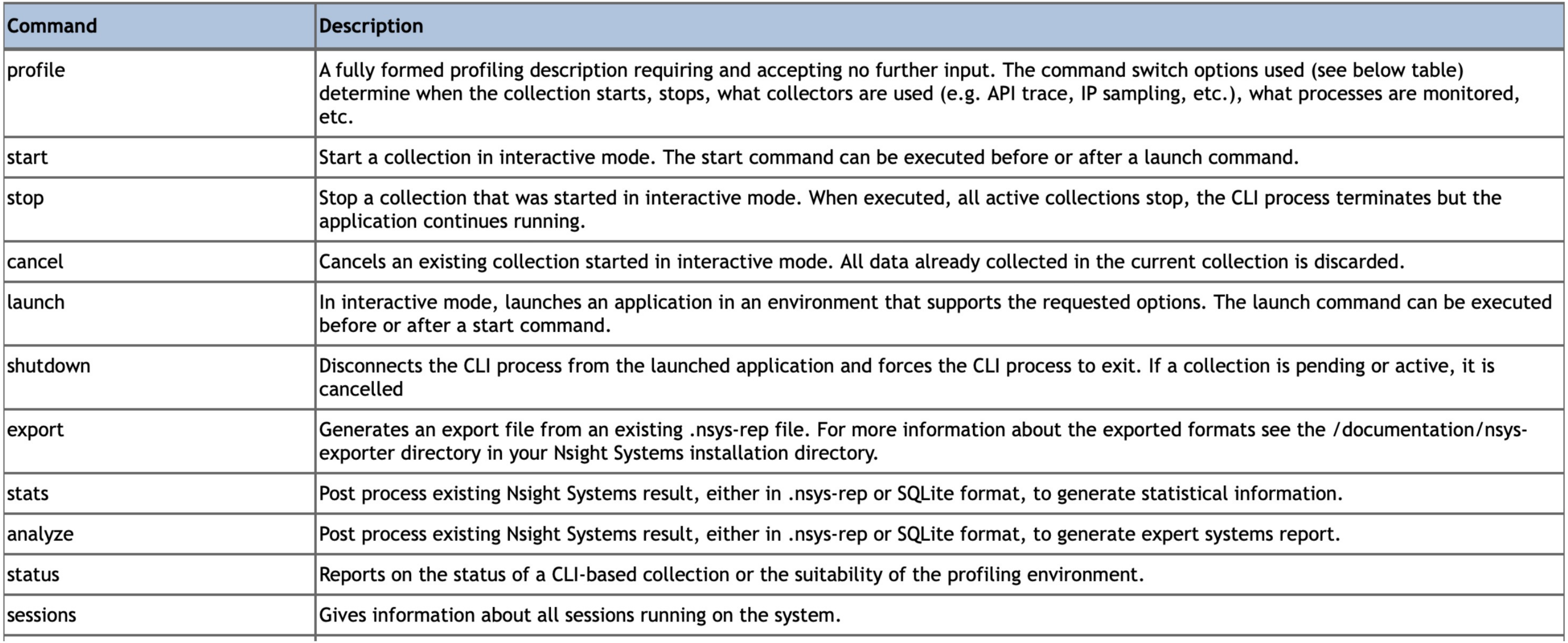
https://developer.nvidia.com/nsight-systems

NVIDIA NSIGHT SYSTEMS

- Support: **MPI**, **OpenACC, OpenMP**
- Complex data mining capabilities, enables to go beyond basic statistics.
- Support multiple simultaneous sessions.
- **MPI trace** feature enables to analyse when the threads are busy or blocked in long-running functions of the **MPI** standard, available on **OpenMPI**, **MPICH** and **NVShmem**.
- **OpenACC** trace enables to see where code has been offload and parallelized onto the GPU, which helps you to analyse the activities executing on the CPUs and GPUs in parallel.
- Tracing **OpenMP** code is available for compilers supporting **OpenMP5** and **OMPT interface**. This capability enables tracing of the parallel regions of code that are distributed either across multiple threads or to the GPU.
- Provides support for **CUDA** graphs. To understand the execution of the source of **CUDA** kernels and execution of **CUDA** graphs, kernels can be correlated back through the graph lunch, instantiation, and all the way back to the code creation, to identify the origin of the kernel execution on the GPU.

https://developer.nvidia.com/nsight-systems

Command Line Options nsys



Command	Description
profile	A fully formed profiling description requiring and accepting no further input. The command switch options used (see below table) determine when the collection starts, stops, what collectors are used (e.g. API trace, IP sampling, etc.), what processes are monitored, etc.
start	Start a collection in interactive mode. The start command can be executed before or after a launch command.
stop	Stop a collection that was started in interactive mode. When executed, all active collections stop, the CLI process terminates but the application continues running.
cancel	Cancels an existing collection started in interactive mode. All data already collected in the current collection is discarded.
launch	In interactive mode, launches an application in an environment that supports the requested options. The launch command can be executed before or after a start command.
shutdown	Disconnects the CLI process from the launched application and forces the CLI process to exit. If a collection is pending or active, it is cancelled
export	Generates an export file from an existing .nsys-rep file. For more information about the exported formats see the /documentation/nsys-exporter directory in your Nsight Systems installation directory.
stats	Post process existing Nsight Systems result, either in .nsys-rep or SQLite format, to generate statistical information.
analyze	Post process existing Nsight Systems result, either in .nsys-rep or SQLite format, to generate expert systems report.
status	Reports on the status of a CLI-based collection or the suitability of the profiling environment.
sessions	Gives information about all sessions running on the system.

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

NSIGHT COMPUTE (ncu)

Interactive CUDA Kernel profiler

- Targeted metric sections for various performance aspects (Debug/&Profile)
- API debugging via a user interface command line tool

- Very high freq. GPU perf counter, customizable data collection and presentation (tables, charts ..,)

- Python-based rules for guided analysis (or postprocessing)

- Provides a customizable and data-driven user interface and metric collection and can be extended with analysis scripts for post-processing results.

https://docs.nvidia.com/nsight-compute/NsightCompute/index.html

Nsight Compute Feature Spotlight in CUDA Toolkit 11 and A100

- **Roofline Analysis**

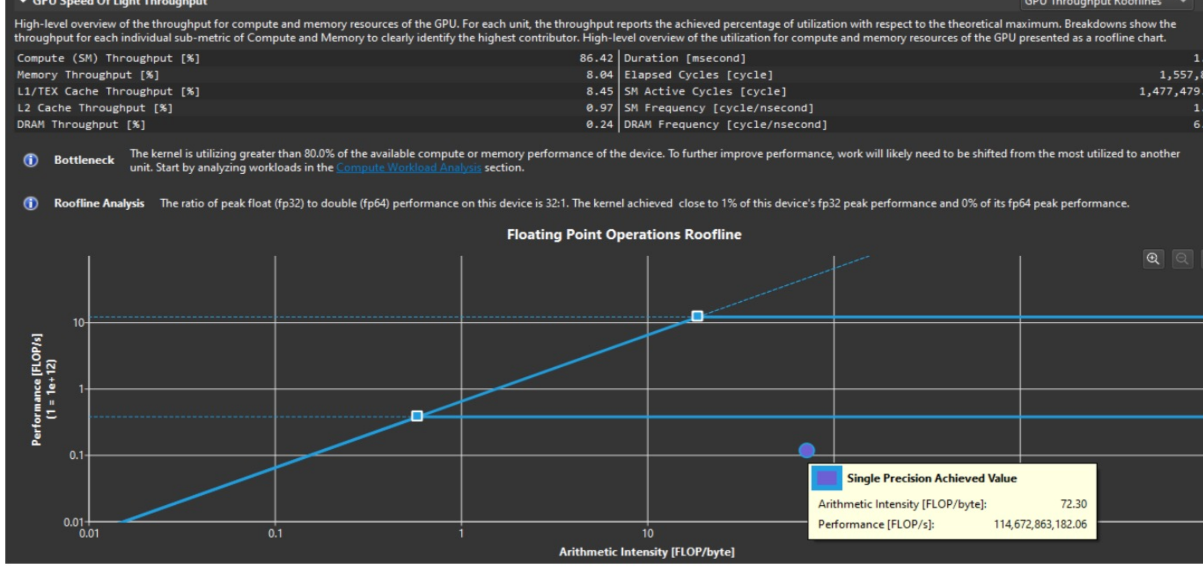
Arithmetic intensity= Compute/Memory

FLOPS = Floating Points Ops/Second

- **Asynchronous copy**

- **Sparse Data Compression**

Shows the amount of data compressed through this feature and the compression ratio, helps on kernels with bandwidth or cache issues.



GPU Speed Of Light Throughput

GPU Throughput Rooflines

High-level overview of the throughput for compute and memory resources of the GPU. For each unit, the throughput reports the achieved percentage of utilization with respect to the theoretical maximum. Breakdowns show the throughput for each individual sub-metric of Compute and Memory to clearly identify the highest contributor. High-level overview of the utilization for compute and memory resources of the GPU presented as a roofline chart.

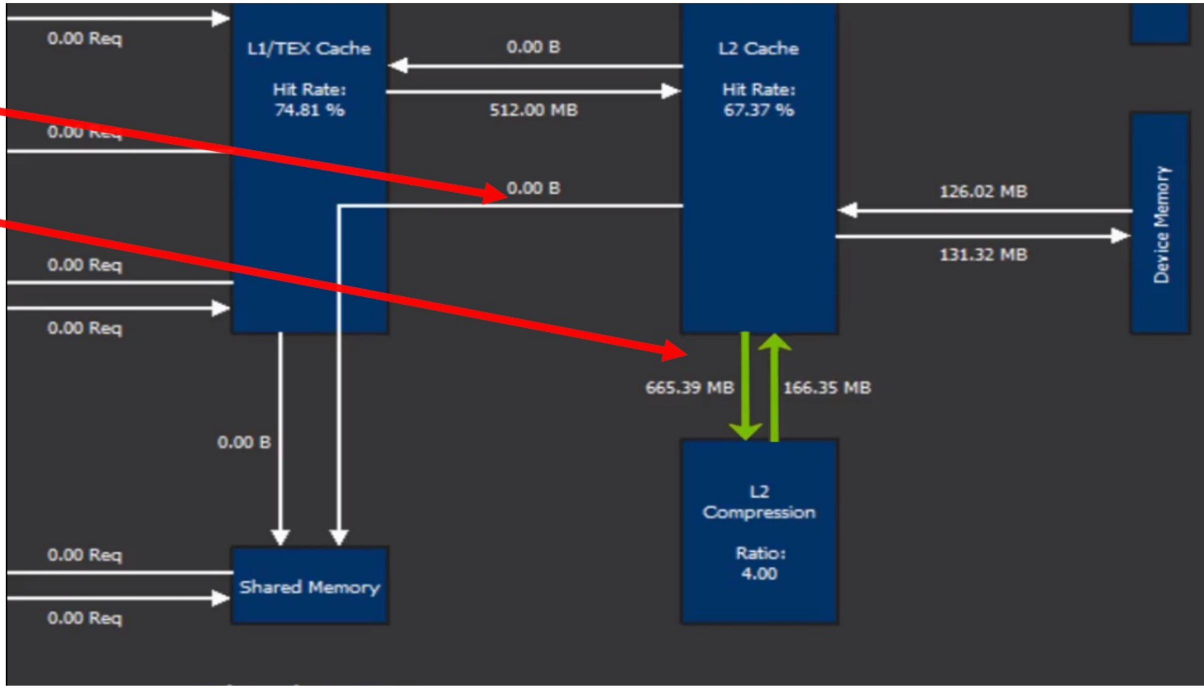
Compute (SM) Throughput [%]	86.42	Duration [msecond]	1
Memory Throughput [%]	8.04	Elapsed Cycles [cycle]	1,557,
L1/TEX Cache Throughput [%]	8.45	SM Active Cycles [cycle]	1,477,479
L2 Cache Throughput [%]	0.97	SM Frequency [cycle/nsecond]	1
DRAM Throughput [%]	0.24	DRAM Frequency [cycle/nsecond]	6

Bottleneck The kernel is utilizing greater than 80.0% of the available compute or memory performance of the device. To further improve performance, work will likely need to be shifted from the most utilized to another unit. Start by analyzing workloads in the Compute Workload Analysis section.

Roofline Analysis The ratio of peak float (fp32) to double (fp64) performance on this device is 32:1. The kernel achieved close to 1% of this device's fp32 peak performance and 0% of its fp64 peak performance.

Floating Point Operations Roofline

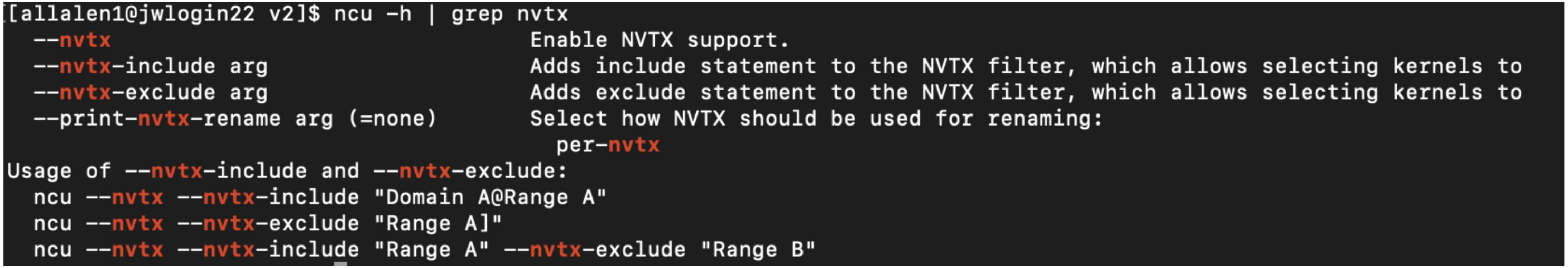
Performance [FLOP/s] (1 = 1e+12)
10
1
0.1
0.01
0.01
0.1
1
10
Arithmetic Intensity [FLOP/byte]
Single Precision Achieved Value
Arithmetic Intensity [FLOP/byte]: 72.30
Performance [FLOP/s]: 114,672,863,182.06



Docs/product: https://developer.nvidia.com/nsight-compute

NVIDIA® Tools Extension SDK (NVTX)

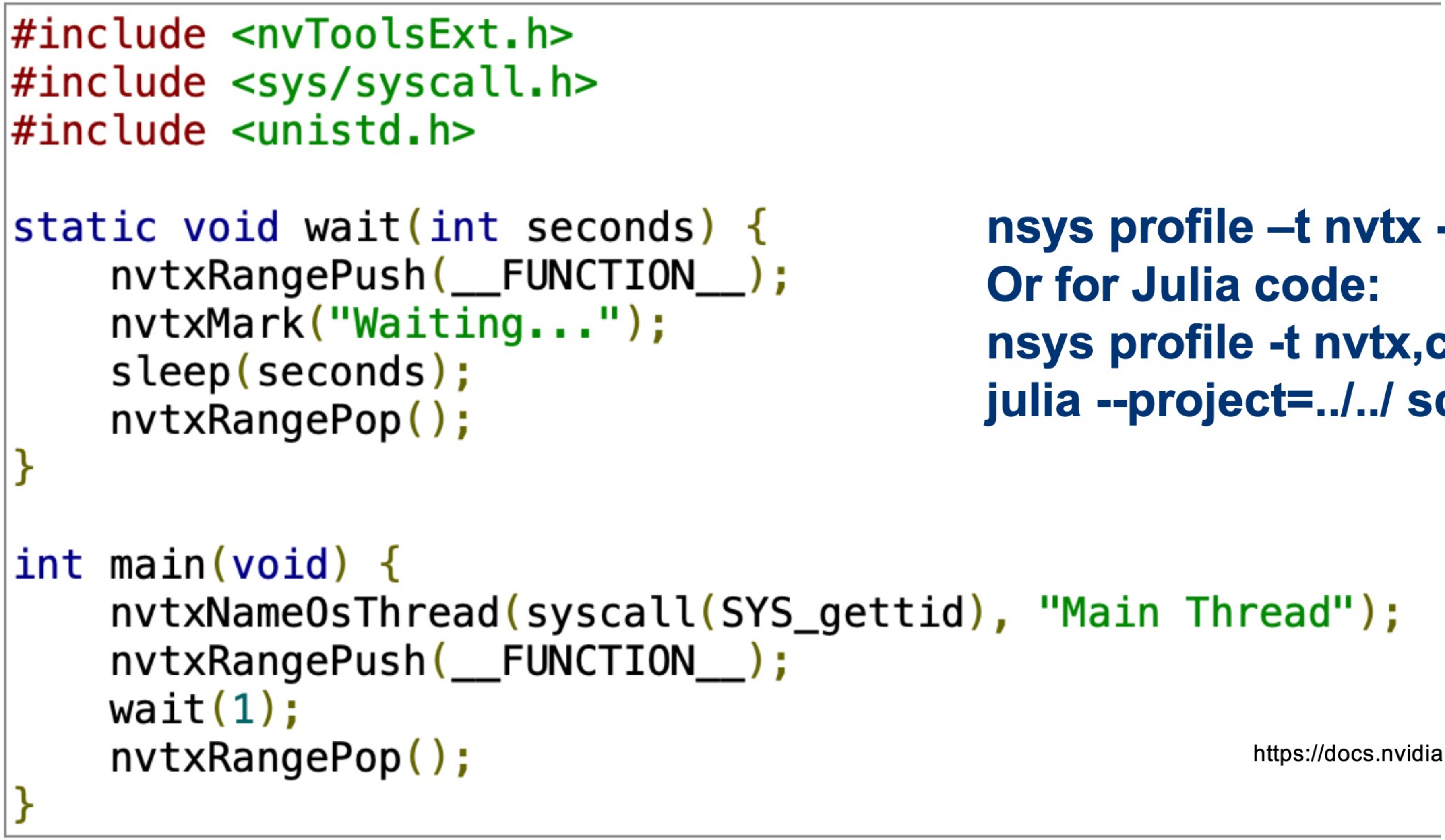
- C-based Application Programming Interface (API) for annotating events, code ranges, and resources in your applications
- Codes which integrate NVTX can use NVIDIA Nsight, Tegra System Profiler, and Visual Profiler to capture and visualize these events and ranges.



```
[allalen1@jwlogin22 v2]$ ncu -h | grep nvtx
  --nvtx                              Enable NVTX support.
  --nvtx-include arg                  Adds include statement to the NVTX filter, which allows selecting kernels to
  --nvtx-exclude arg                  Adds exclude statement to the NVTX filter, which allows selecting kernels to
  --print-nvtx-rename arg (=none)     Select how NVTX should be used for renaming:
                                        per-nvtx
Usage of --nvtx-include and --nvtx-exclude:
  ncu --nvtx --nvtx-include "Domain A@Range A"
  ncu --nvtx --nvtx-exclude "Range A]"
  ncu --nvtx --nvtx-include "Range A" --nvtx-exclude "Range B"
```

https://docs.nvidia.com/nsight-visual-studio-edition/nvtx/index.html

NVIDIA® Tools Extension SDK (NVTX)



```
#include <nvToolsExt.h>
#include <sys/syscall.h>
#include <unistd.h>

static void wait(int seconds) {
    nvtxRangePush(__FUNCTION__);
    nvtxMark("Waiting...");
    sleep(seconds);
    nvtxRangePop();
}

int main(void) {
    nvtxNameOsThread(syscall(SYS_gettid), "Main Thread");
    nvtxRangePush(__FUNCTION__);
    wait(1);
    nvtxRangePop();
}
```

nsys profile –t nvtx --stats=true …

Or for Julia code:

nsys profile -t nvtx,cuda -o output_file.qdrep

julia --project=../../ script.jl

https://docs.nvidia.com/nsight-visual-studio-edition/2020.1/nvtx/index.html

A First (I)Nsight

Recording with the CLI

- 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`

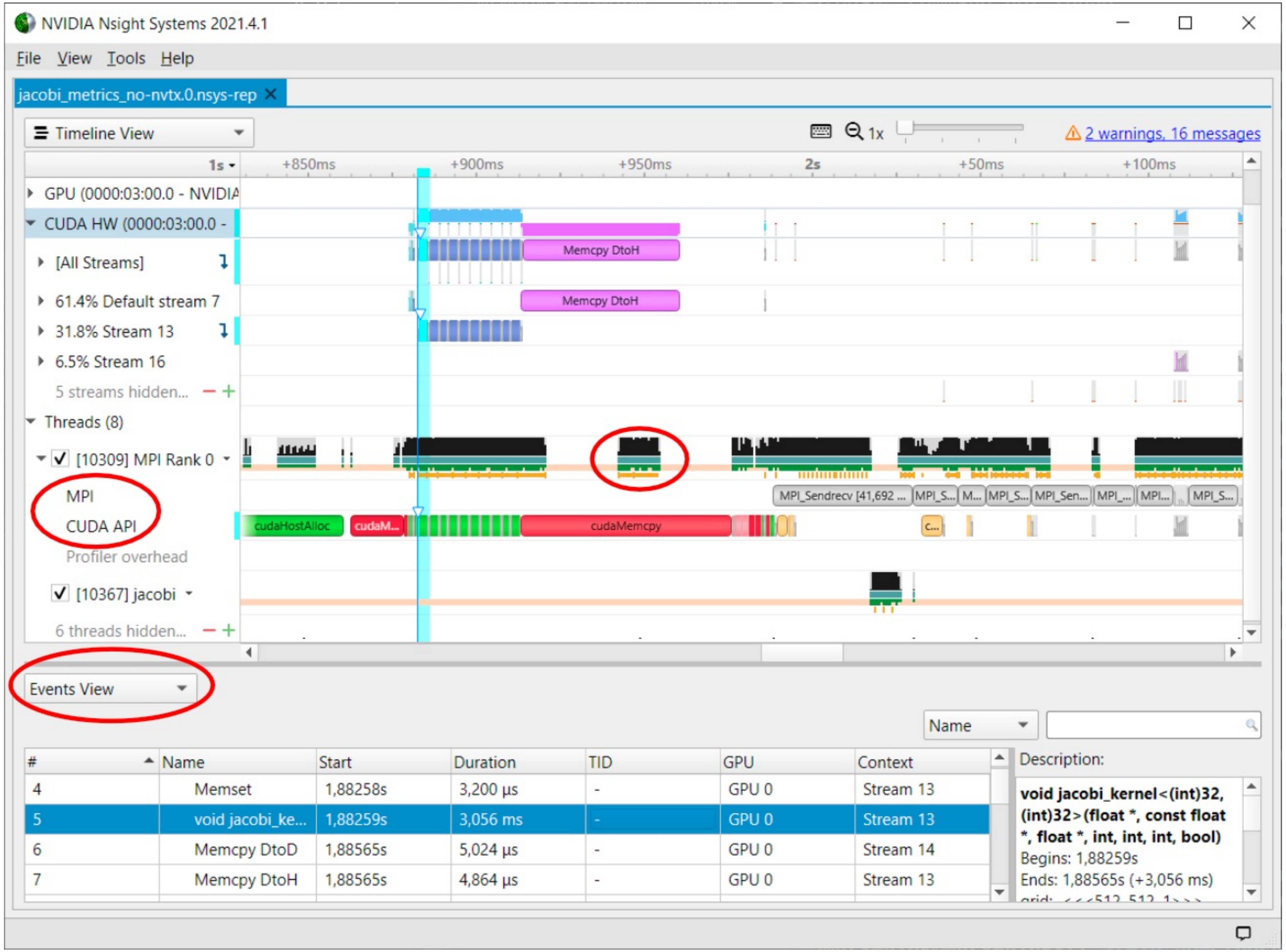
- Runs set of reports on command line, customizable (**sqlite** + `Python`):
  - Useful to check validity of profile, identify important kernels

```
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
```

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
  - Expert Systems
- looks at single process (tree)
  - correlate multi-process reports in single timeline



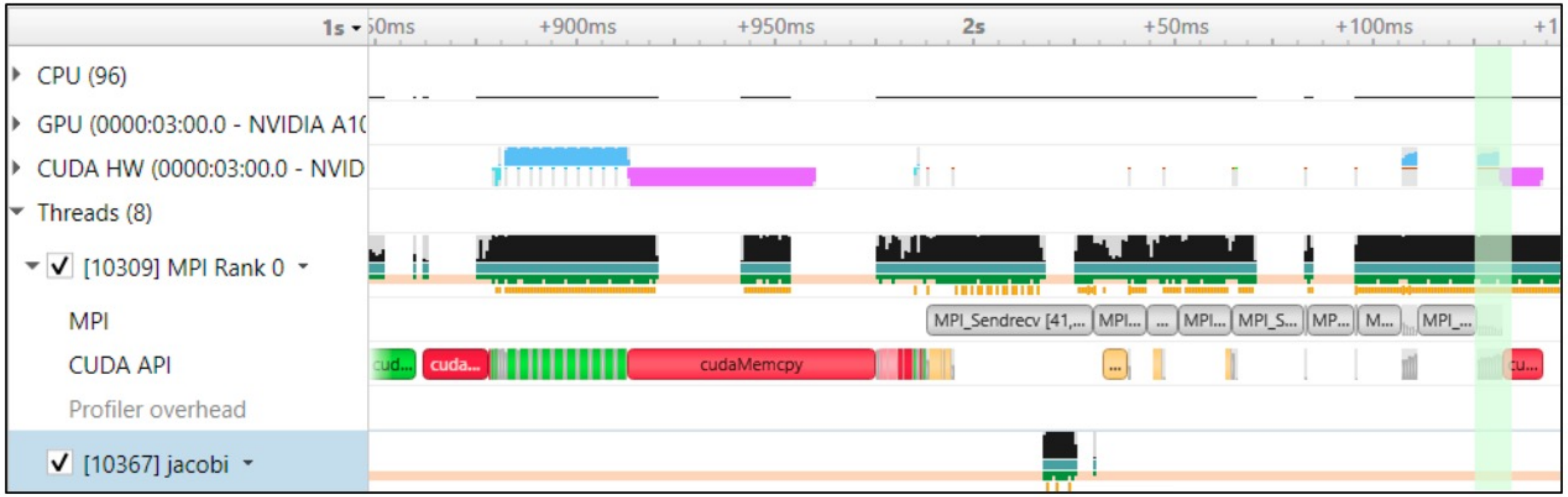
#	Name	Start	Duration	TID	GPU	Context
4	Memset	1,88258s	3,200 µs	-	GPU 0	Stream 13
5	void jacobi_ke...	1,88259s	3,056 ms	-	GPU 0	Stream 13
6	Memcpy DtoD	1,88565s	5,024 µs	-	GPU 0	Stream 14
7	Memcpy DtoH	1,88565s	4,864 µs	-	GPU 0	Stream 13

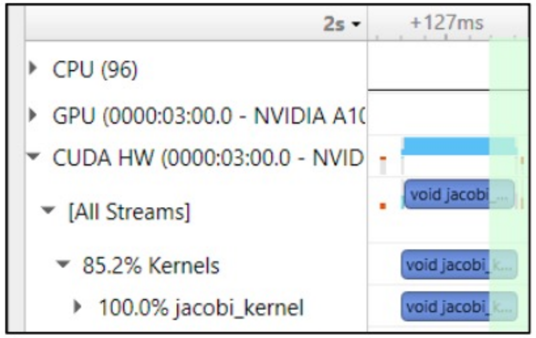
Description:

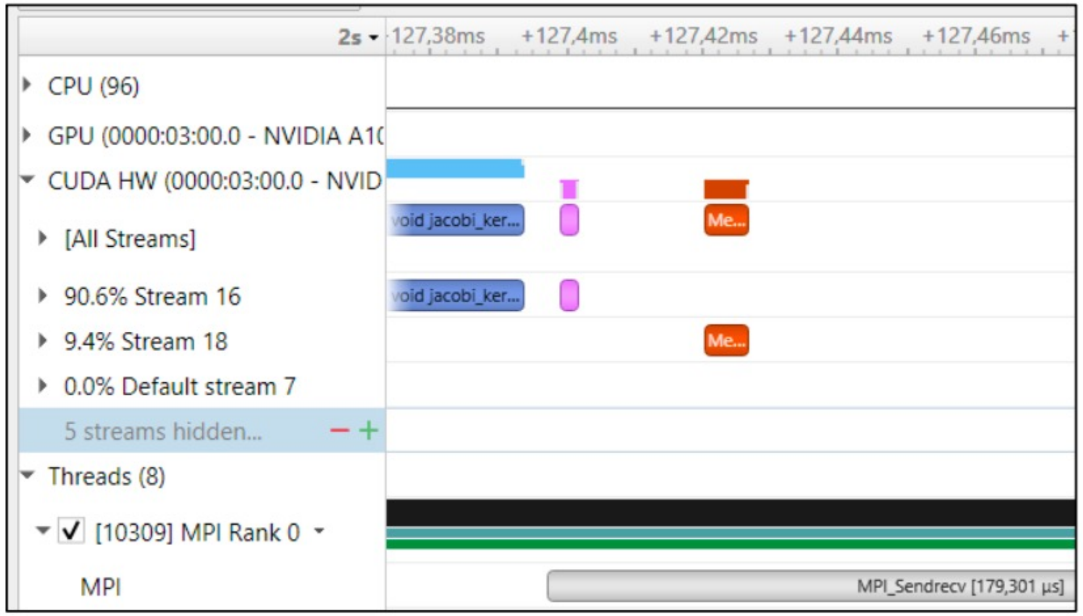
void jacobi_kernel<(int)32, (int)32>(float *, const float *, float *, int, int, int, bool)
Begins: 1,88259s
Ends: 1,88565s (+3,056 ms)
grid: <<<512, 512, 1>>>

Discovering Optimization Potential

- Using Jacobi solver example*
- Spot kernels – lots of whitespace
  - Which part is „bad“?
  - Enhance!
- MPI calls
  - Memory copies
  - We know: This is CUDA-aware MPI
- Even without knowing source, insight
- Too complicated for repeated/reliable usage
  - How to simplify navigating and comparing reports?







*See https://github.com/NVIDIA/multi-gpu-programming-models/

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

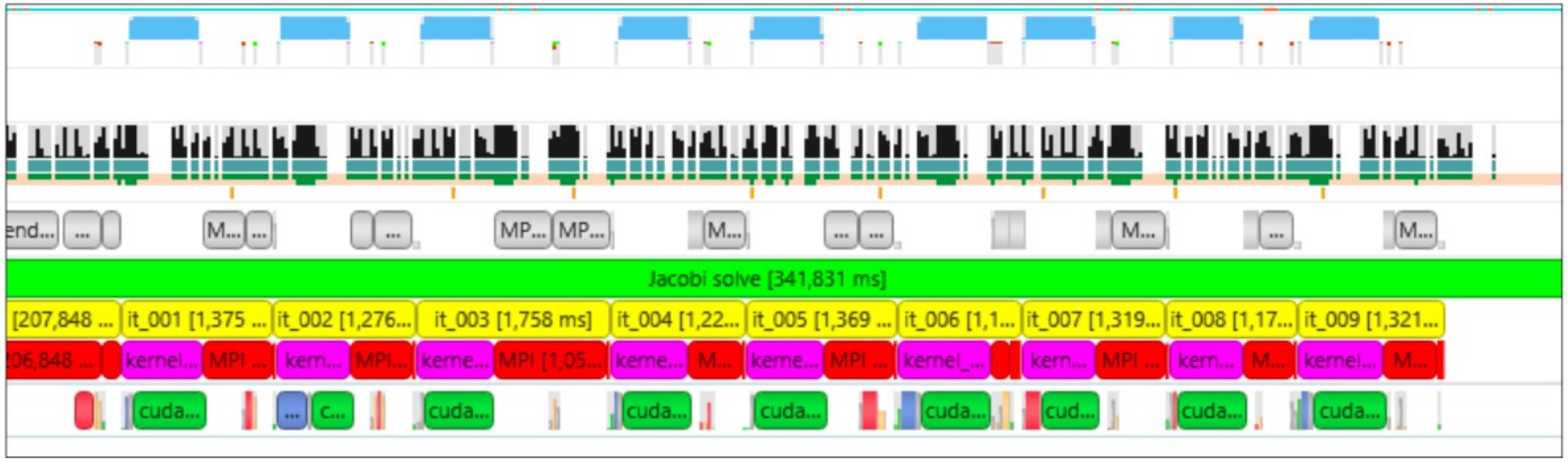
https://developer.nvidia.com/blog/cuda-pro-tip-generate-custom-application-profile-timelines-nvtx/
https://developer.nvidia.com/blog/customize-cuda-fortran-profiling-nvtx/

Minimizing Profile Size

Shorter time, smaller files = quicker progress

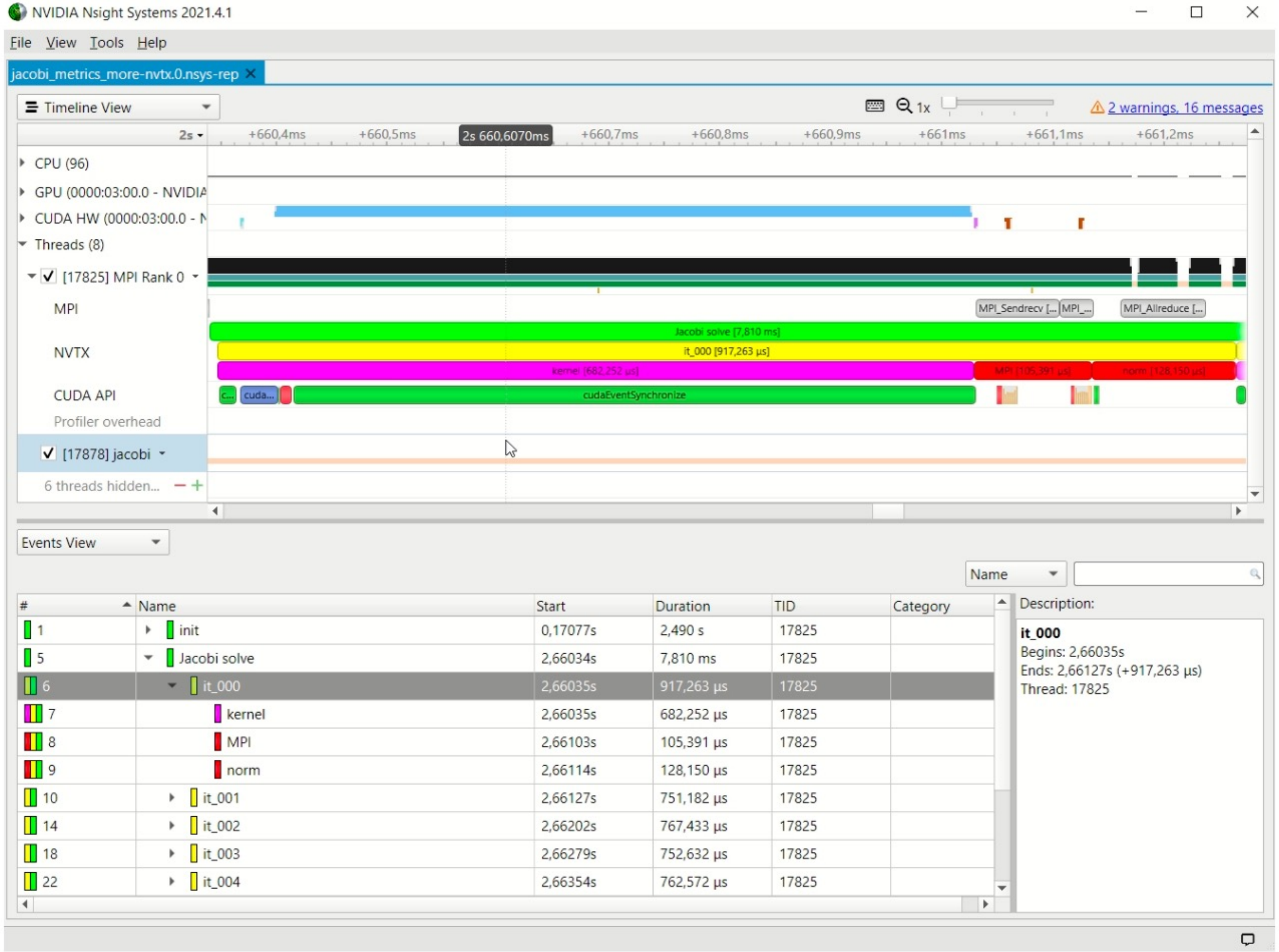
- Only profile what you need – all profilers have some overhead
  - Example: Event that occurs after long-running setup phase
- Bonus: lower number of events leads to smaller file size
- Add to nsys command line:

  -
    ```
    --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`



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	

Description:

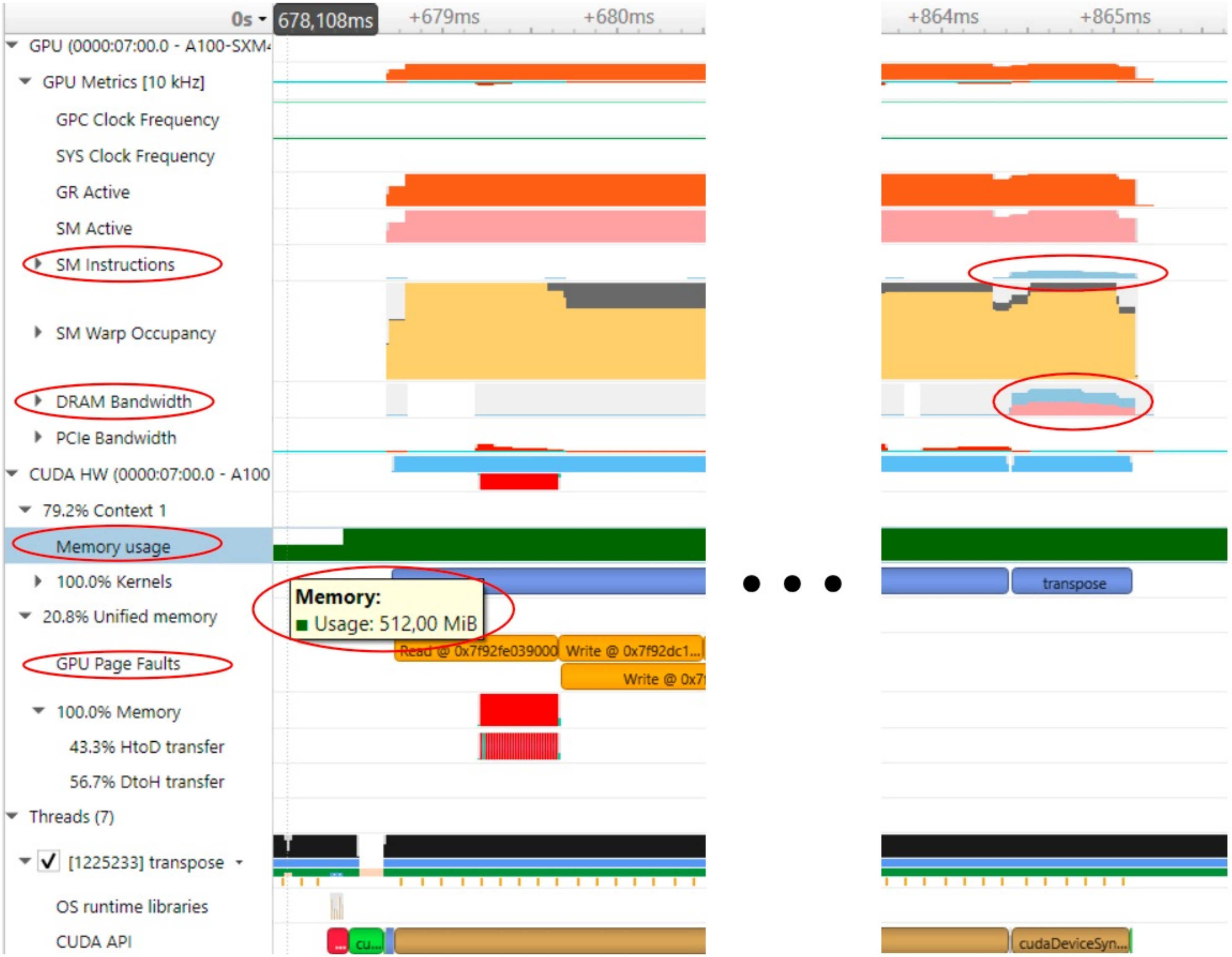
it_000
Begins: 2,66035s
Ends: 2,66127s (+917,263 µs)
Thread: 17825

GPU Metrics in Nsight Systems

...and other traces you can activate

- Valuable low-overhead insight into HW usage:
  - SM instructions
  - DRAM Bandwidth, PCIe Bandwith (GPUDirect)
- Also: Memory usage, Page Faults (higher overhead)
  - CUDA Programming guide: **Unified Memory Programming**
- Can save kernel-level profiling effort!

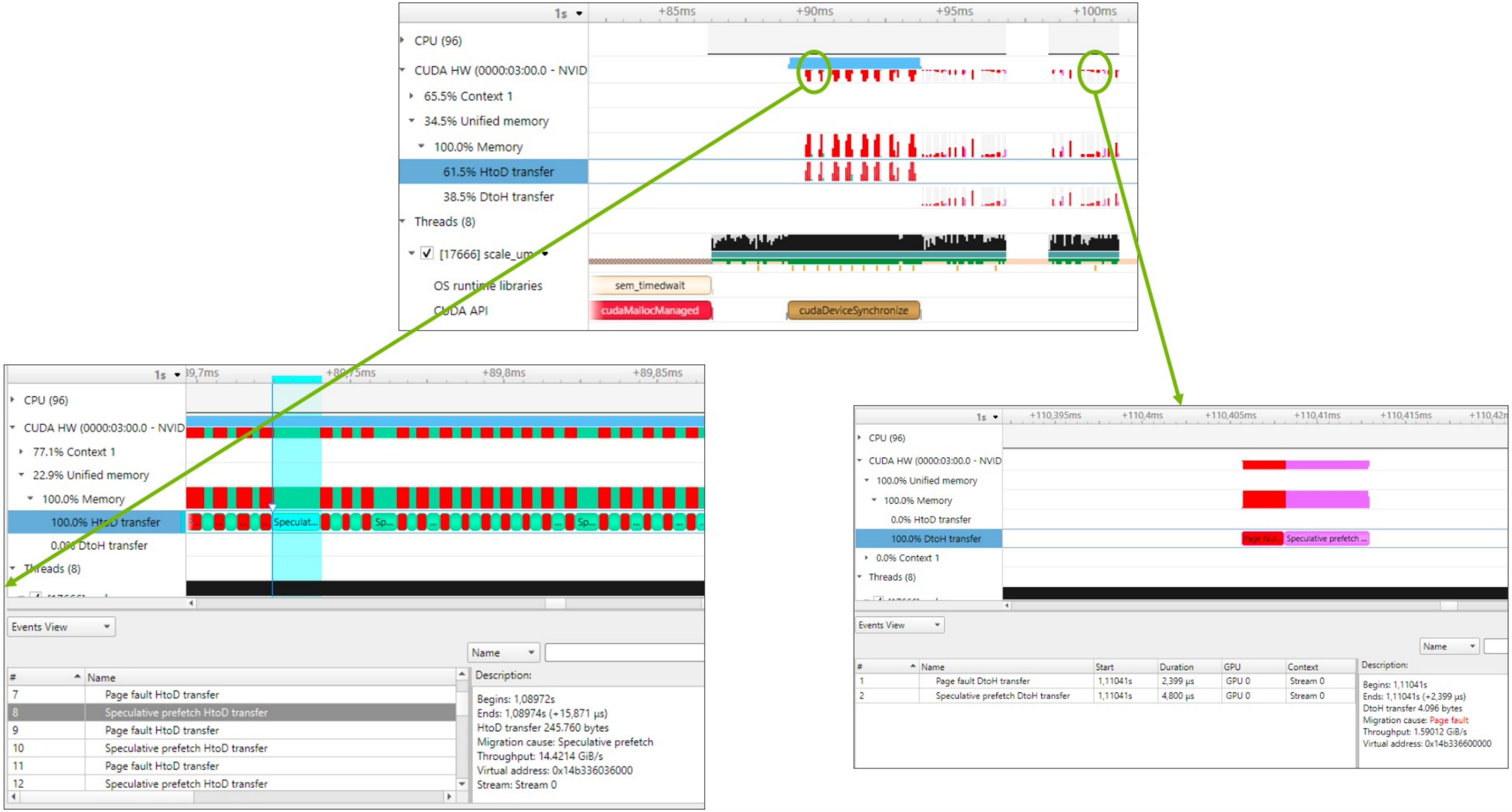
-
  ```
  nsys profile
  --gpu-metrics-device=0
  --cuda-memory-usage=true
  --cuda-um-cpu-page-faults=true
  --cuda-um-gpu-page-faults=true
  ./app
  ```



Memory:
🟩 Usage: 512,00 MiB

Unified Memory movement

Observing transfers in Nsight Systems



#	Name
7	Page fault HtoD transfer
8	Speculative prefetch HtoD transfer
9	Page fault HtoD transfer
10	Speculative prefetch HtoD transfer
11	Page fault HtoD transfer
12	Speculative prefetch HtoD transfer

Description:

Begins: 1,08972s
Ends: 1,08974s (+15,871 µs)
HtoD transfer 245.760 bytes
Migration cause: Speculative prefetch
Throughput: 14.4214 GiB/s
Virtual address: 0x14b336036000
Stream: Stream 0

#	Name	Start	Duration	GPU	Context
1	Page fault DtoH transfer	1,11041s	2,399 µs	GPU 0	Stream 0
2	Speculative prefetch DtoH transfer	1,11041s	4,800 µs	GPU 0	Stream 0

Description:

Begins: 1,11041s
Ends: 1,11041s (+2,399 µs)
DtoH transfer 4.096 bytes
Migration cause: Page fault
Throughput: 1.59012 GiB/s
Virtual address: 0x14b336600000

NSIGHT SYSTEMS

System-wide application algorithm tuning

- Multi-process tree support

Locate optimization opportunities

- Visualize millions of events on a very fast GUI timeline
- Or gaps of unused CPU and GPU time

Balance your workload across multiple CPUs and GPUs

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

Multi-platform: Linux & Windows, x86-64, Tegra, Power, MacOSX(host only)

GPUs: Volta, Turing

Docs/product: https://developer.nvidia.com/nsight-systems

NSIGHT COMPUTE

CUDA Kernel profiler

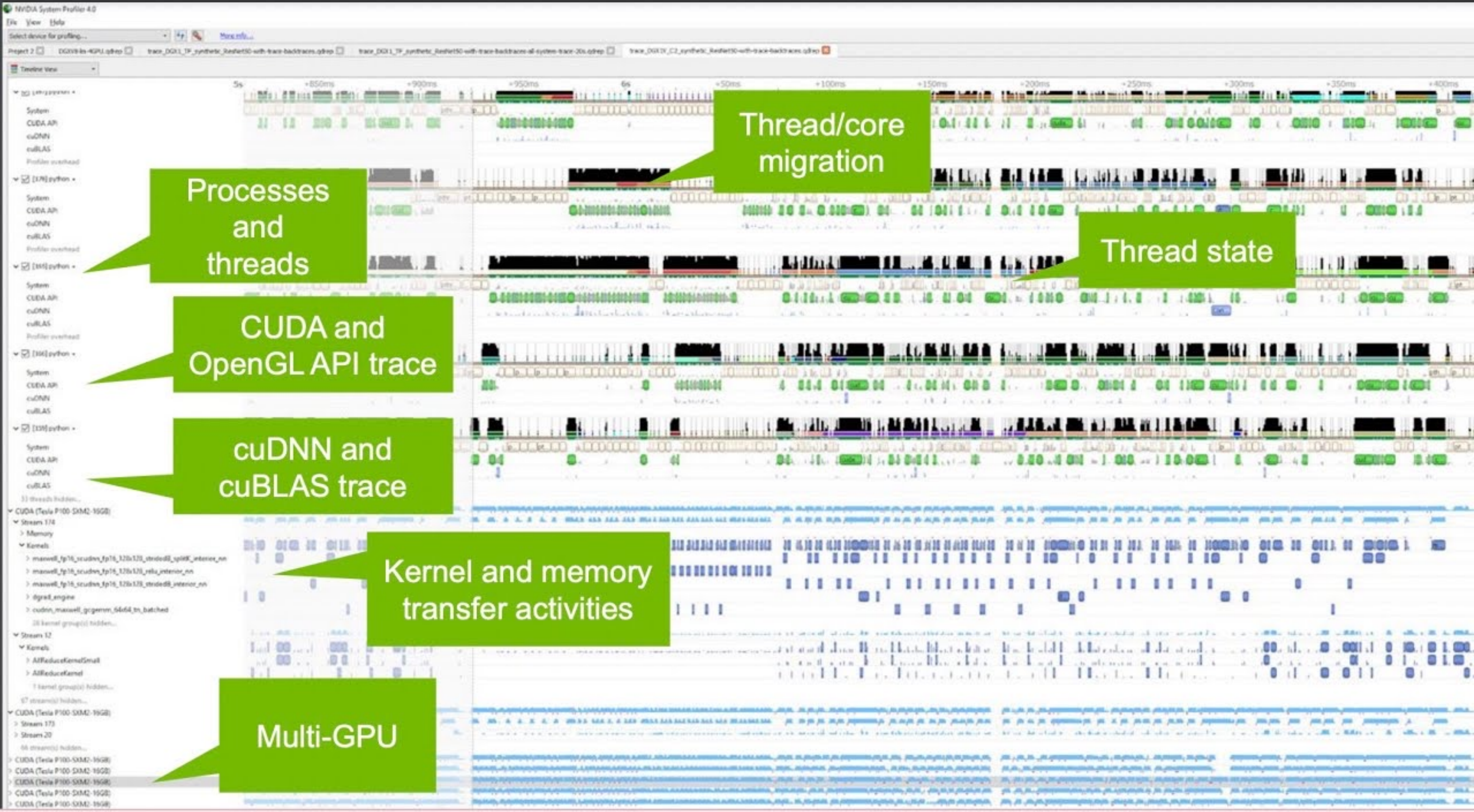
Targeted metric sections for various performance aspects (Debug/&Profile)

Very high freq GPU perf counter, customizable data collection and presentation (tables, charts ..,)

Python-based rules for guided analysis (or postprocessing)

GPUs: Volta, Turing, Amper…

Docs/product: https://developer.nvidia.com/nsight-systems


NVIDIA System Profiler 4.0
File View Help
Select device for profiling...
More info...
Project 2
DGXVB-lin-4GPU.qdrep
trace_DGX1_TP_synthetic_ResNet50-with-trace-backtraces.qdrep
trace_DGX1_TP_synthetic_ResNet50-with-trace-backtraces-all-system-trace-20s.qdrep
trace_DGX1V_C2_synthetic_ResNet50-with-trace-backtraces.qdrep
Timeline View
5s
+850ms
+900ms
+950ms
6s
+50ms
+100ms
+150ms
+200ms
+250ms
+300ms
+350ms
+400ms
System
CUDA API
cuDNN
cuBLAS
Profiler overhead
[178] python
[165] python
[166] python
[109] python
33 threads hidden...
CUDA (Tesla P100-SXM2-16GB)
Stream 174
Memory
Kernels
maxwell_fp16_scudnn_fp16_128x128_stridedB_splitK_interior_nn
maxwell_fp16_scudnn_fp16_128x128_relu_interior_nn
maxwell_fp16_scudnn_fp16_128x128_stridedB_interior_nn
dgrad_engine
cudnn_maxwell_gcgemm_64x64_tn_batched
28 kernel group(s) hidden...
Stream 12
Kernels
AllReduceKernelSmall
AllReduceKernel
1 kernel group(s) hidden...
67 stream(s) hidden...
CUDA (Tesla P100-SXM2-16GB)
Stream 173
Stream 20
66 stream(s) hidden...
CUDA (Tesla P100-SXM2-16GB)
CUDA (Tesla P100-SXM2-16GB)
CUDA (Tesla P100-SXM2-16GB)
CUDA (Tesla P100-SXM2-16GB)
CUDA (Tesla P100-SXM2-16GB)

Thread/core migration

Processes and threads

Thread state

CUDA and OpenGL API trace

cuDNN and cuBLAS trace

Kernel and memory transfer activities

Multi-GPU

NVIDIA Tools Extension API Library (NVTX)

The NVIDIA Tools Extension SDK (NVTX) is a C-based Application Programming Interface (API) for annotating events, code ranges, and resources in your applications.
Applications which integrate NVTX can use NVIDIA Nsight VSE to capture and visualize these events and ranges.

```
void Wait(int waitMilliseconds)
{
                nvtxNameOsThread(“MAIN”);
                nvtxRangePush(__FUNCTION__);
                nvtxMark(>"Waiting...");
                Sleep(waitMilliseconds);
                nvtxRangePop();
 }
 int main(void)
 {
                nvtxNameOsThread("MAIN");
                nvtxRangePush(__FUNCTION__);
                Wait();
                nvtxRangePop();
}
```

nsys profile –t nvtx --stats=true …

https://docs.nvidia.com/nsight-visual-studio-edition/2020.1/nvtx/index.html

IS

THANK YOU

Instructor: Dr. Momme Allalen
www.nvidia.com/dli