lrz

Leibniz-Rechenzentrum
der Bayerischen Akademie der Wissenschaften

Introduction into NVIDIA® Nsight™ Systems

HLRS | 11– 13 July 2023

CUDA® PROFILING TOOLS

nvvc: NVIDIA visual profiler
nvprof: tool to understand and optimize the performance of your CUDA, OpenACC or OpenMP applications,

Application level opportunities
- Overall application performance
  - Overlap CPU and GPU work, identify the bottlenecks (CPU or GPU)
- Overall GPU utilization and efficiency
  - Overlap compute and memory copies
  - Utilize compute and copy engines effectively.

Kernel level opportunities
- Use memory bandwidth efficiently
- Use compute resources efficiently
- Hide instruction and memory latency

There are more features, example for Dependency Analysis
Command: **nvprof** --dependency-analysis --cpu-thread-tracing on ./ executable_cuda

Nsight Systems
Nsight Compute

THE NSIGHT SUITE COMPONENTS

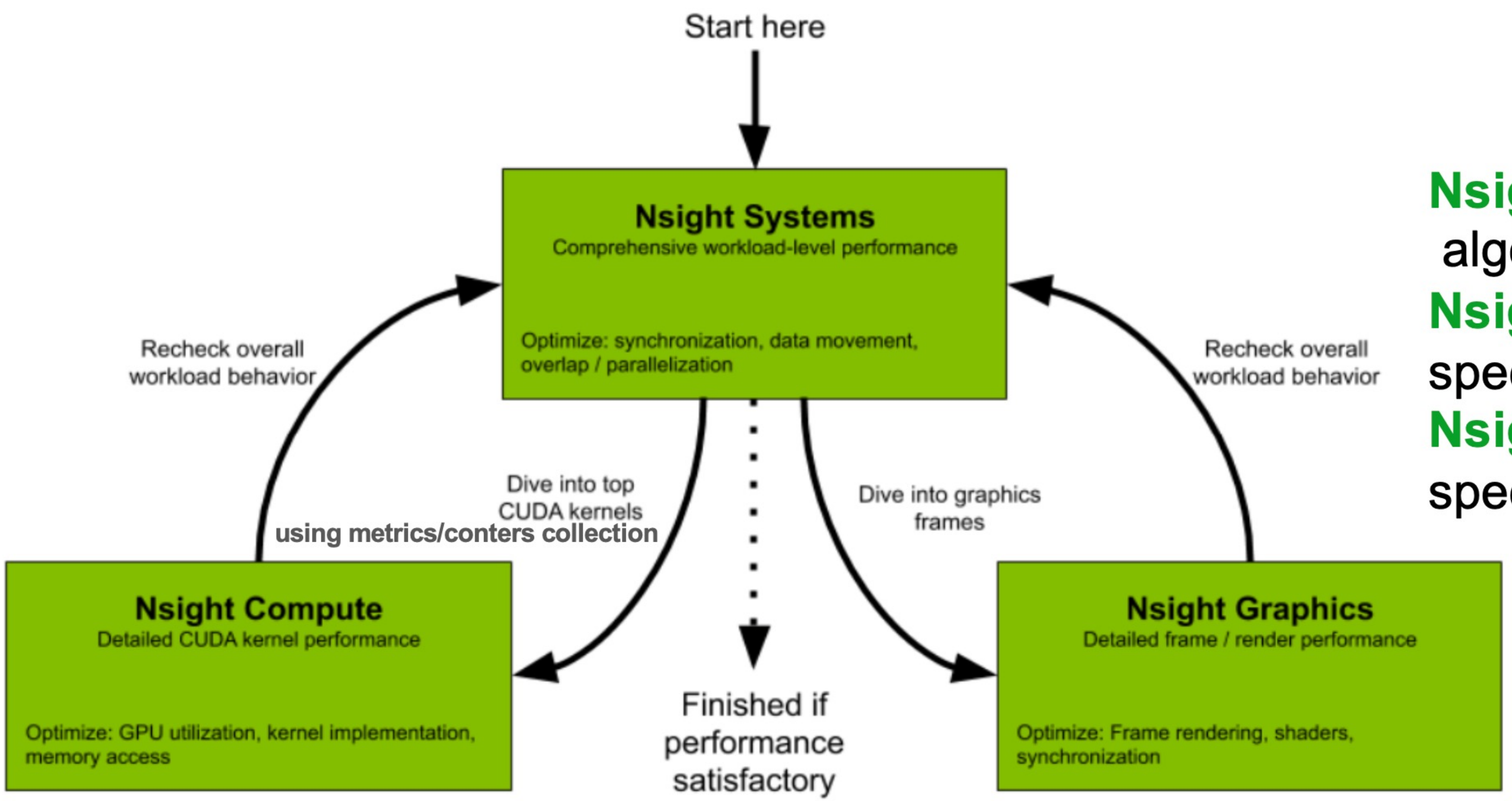


Figure 1. Flowchart describing working with new NVIDIA Nsight tools for performance optimization

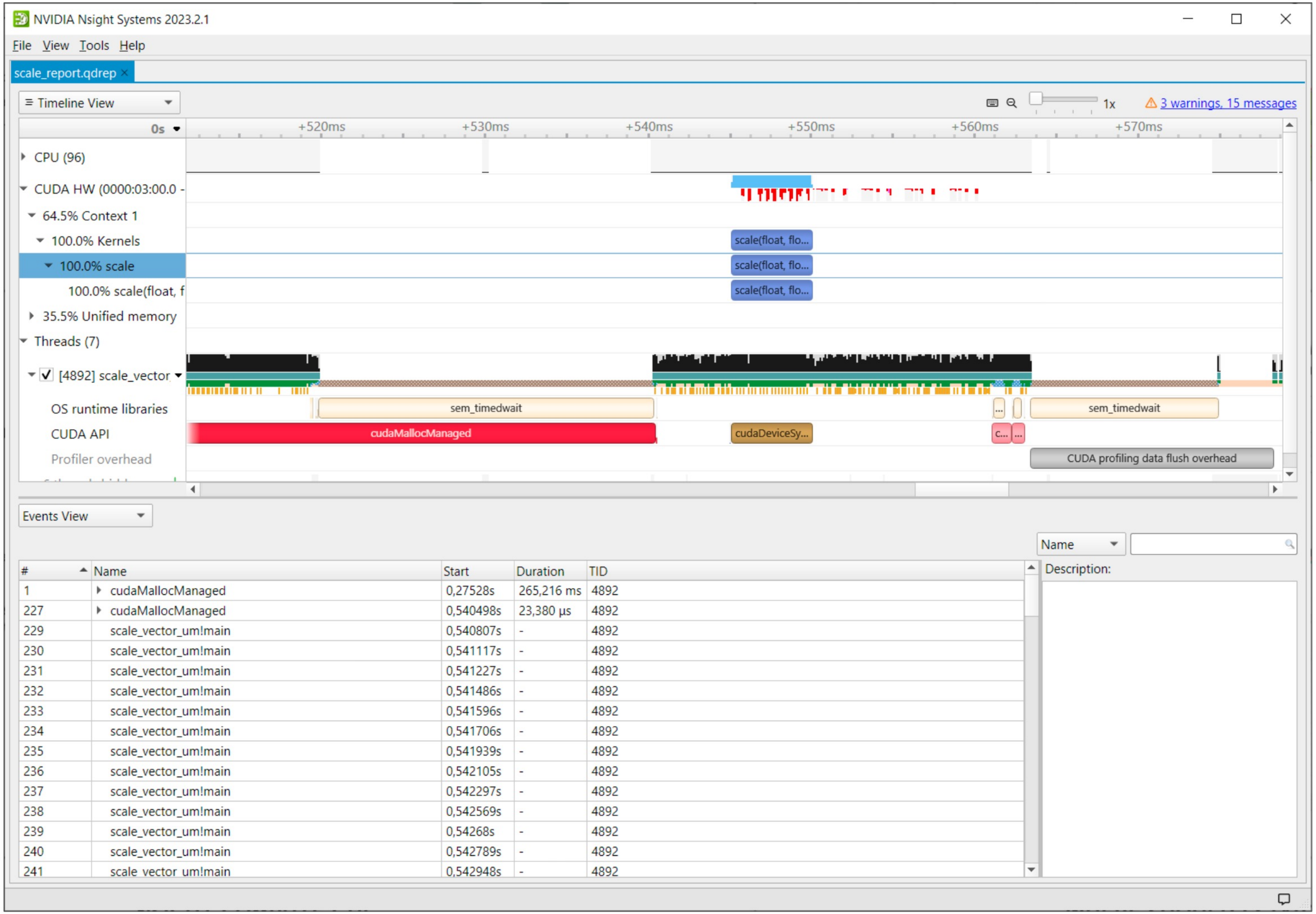
Nsight Systems – Analyze application algorithm system wide
Nsight Compute – Debug/&optomize specific CUDA kernels
Nsight Graphics – Debug/&optimize specific graphics workloads

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

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

Nsight Systems GUI

Main timeline view, Events View



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)

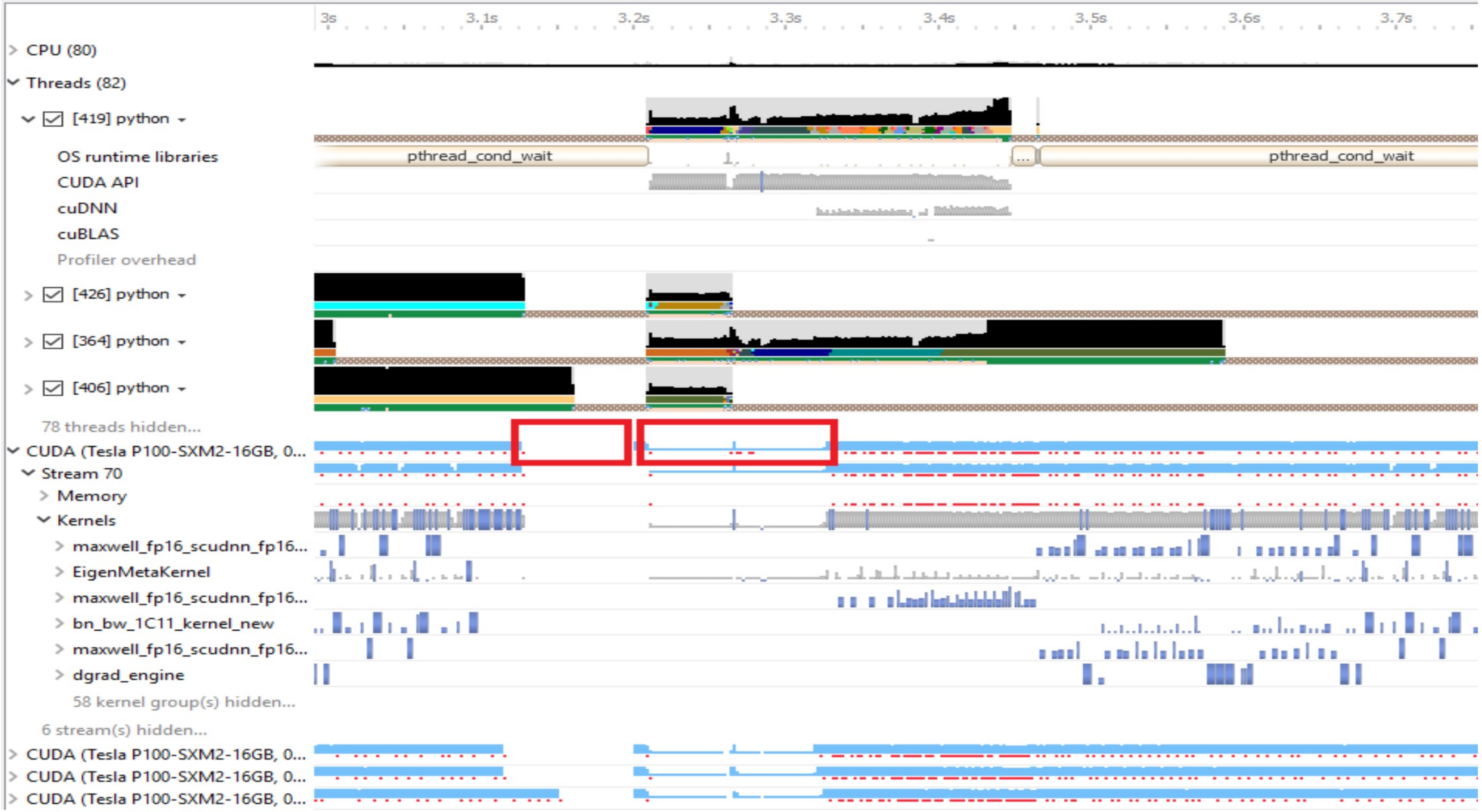
https://developer.nvidia.com/nsight-systems

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.

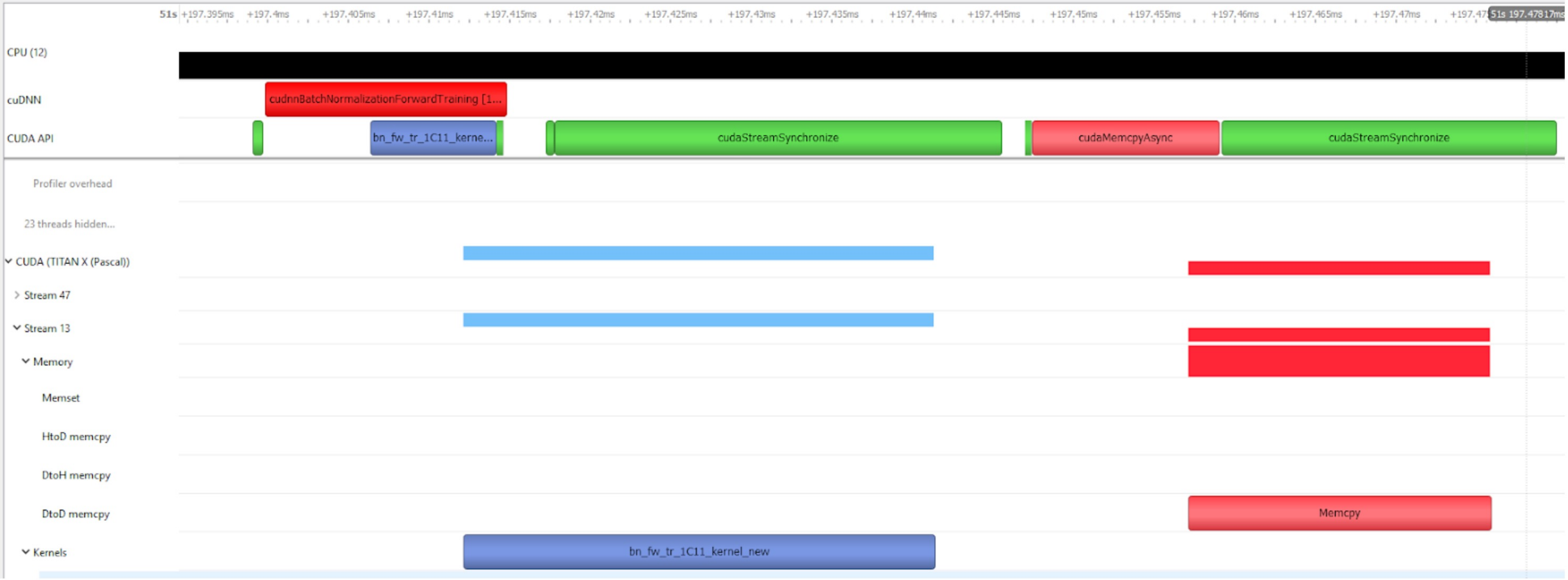
https://docs.nvidia.com/nsight-systems/UserGuide/index.html

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

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

NVIDIA NSIGHT COMPUTE
Important Features

- Result comparison across one or multiple reports within the tool
- Graphical profile report
- Interactive kernel profiler and API debugger: debugging CPU and GPU simultaneously and capable of handling thousands of simultaneous threads.
- Fast data collection
- GUI and command line interface
- Fully customizable reports and analysis rules

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

Nsight Compute Feature Spotlight in CUDA Toolkit 11 and A100

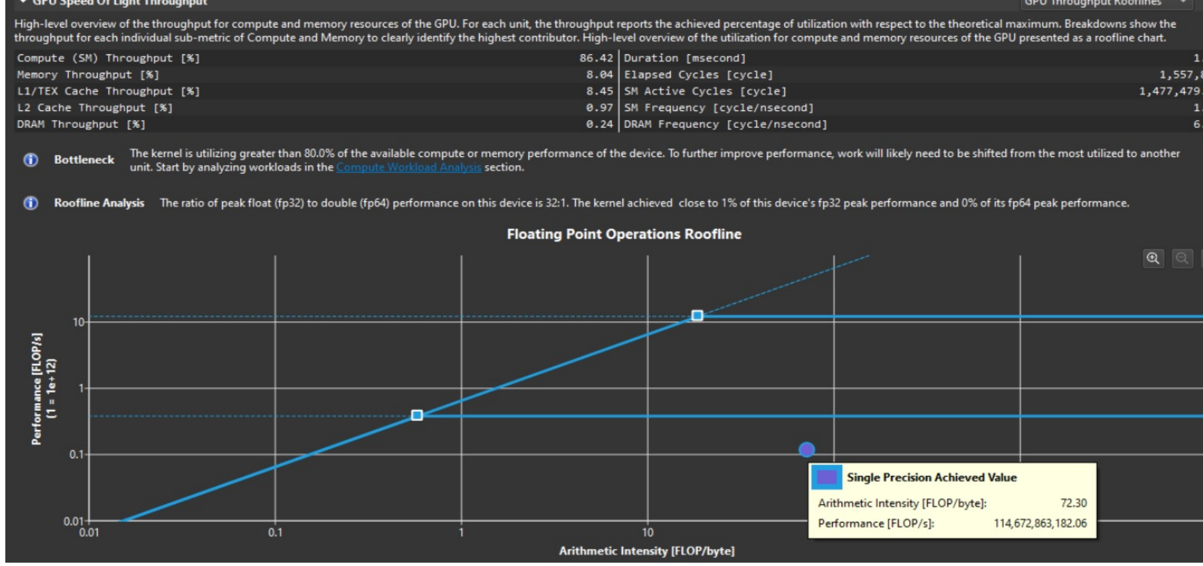
- **Roofline Analysis**

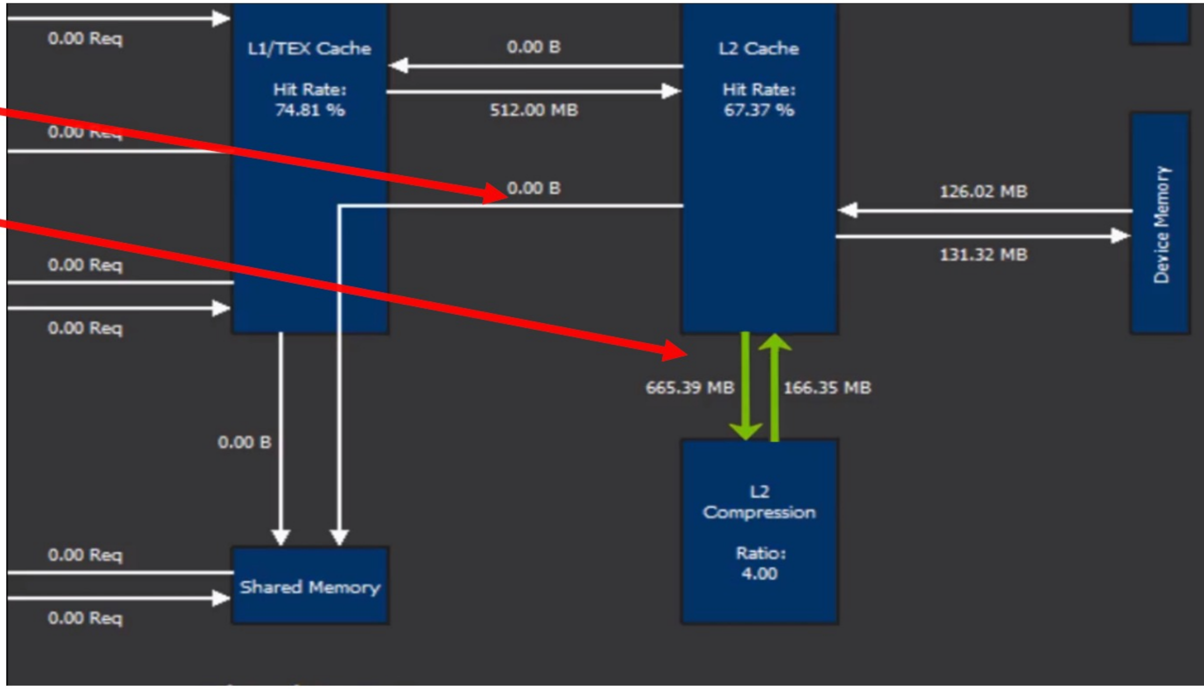
  Arithmetic intensity= Compute/Memory
  FLOPS = Floating Points Ops/Second

- **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.





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

NVIDIA® Tools Extension SDK (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();
}
```

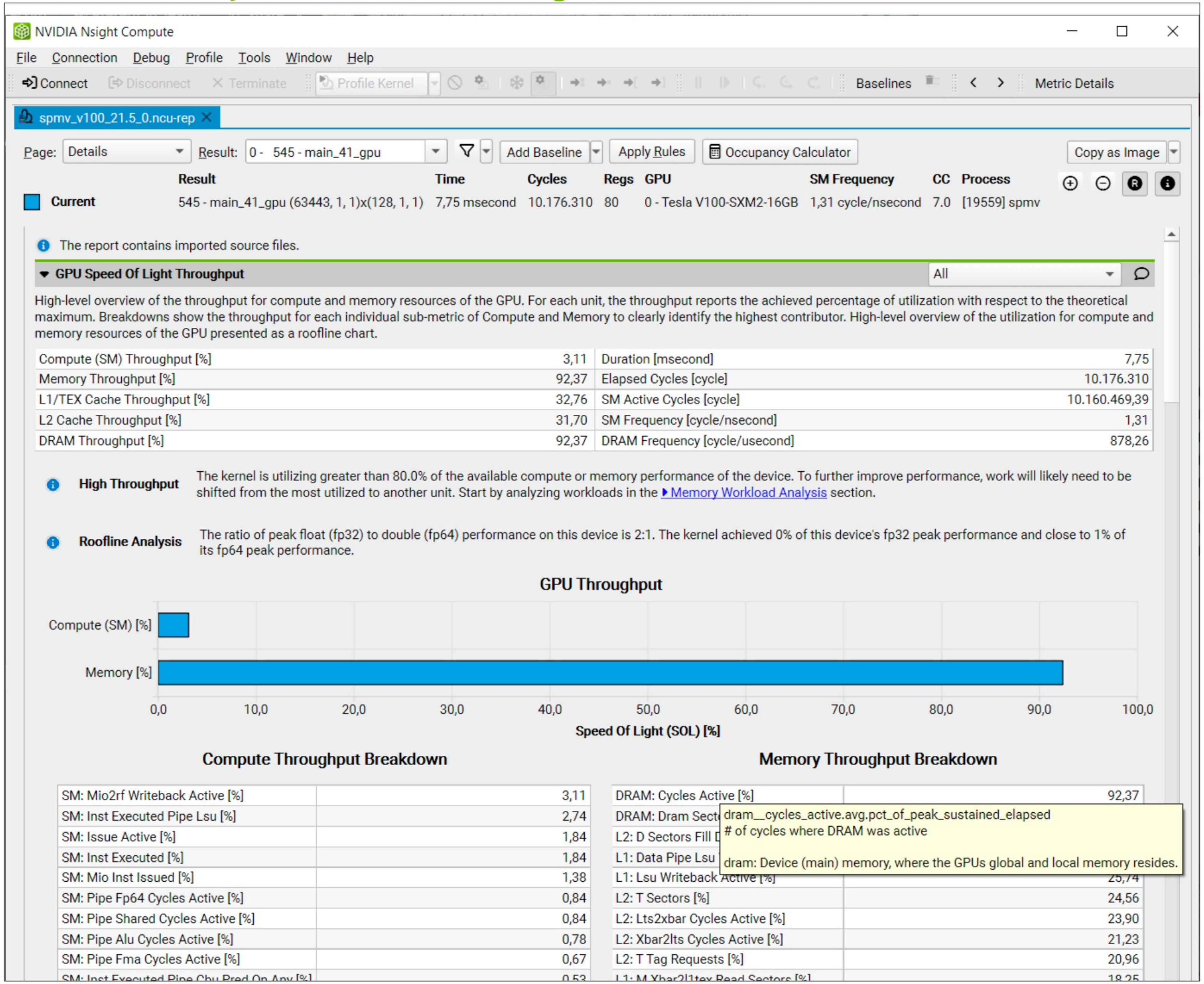
nsys profile –t nvtx --stats=true …

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

Nsight Compute GUI

First steps in kernel analysis - Understanding the initial limiter

- GPU "Speed of Light Throughput"
  - SOL = theoretical peak
- "Breakdown" tables
  - DRAM: Cycles Active
- Tooltips
- Rules point to next steps



https://docs.nvidia.com/nsight-compute/NsightCompute/index.html?ncid=em-prod-821317#cid=dev02_em-prod_en-us

De

A First (I)Nsight

Recording with the CLI

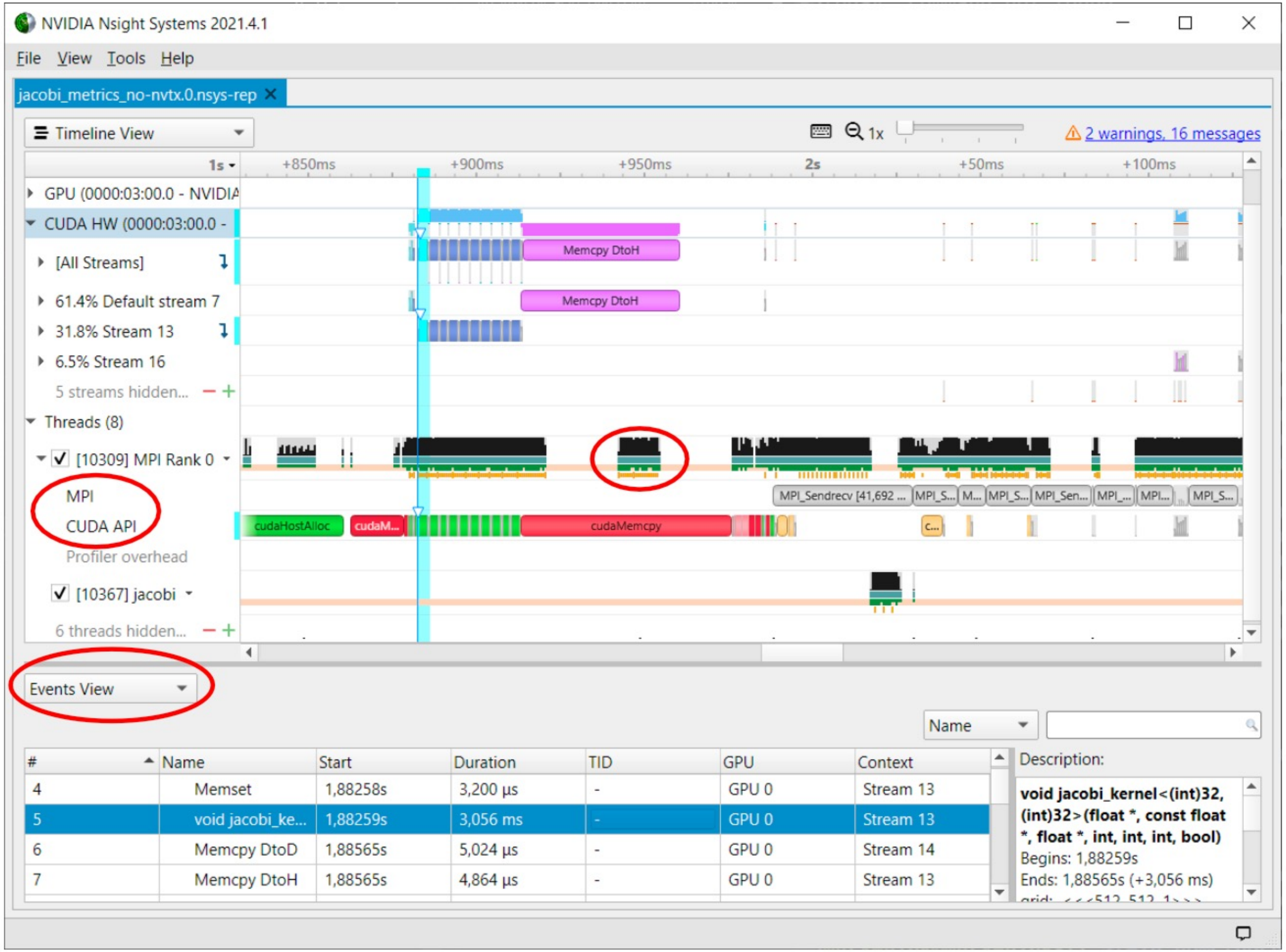
- Use the command line
  - `srun nsys profile --trace=cuda,nvtx,mpi --force-overwrite=true --output=my_report.%q{SLURM_PROCID} \ ./jacobi -niter 10`
- Inspect results: Open the report file in the GUI
  - Also possible to get details on command line
  - Either add `--stats` to profile command line, or: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

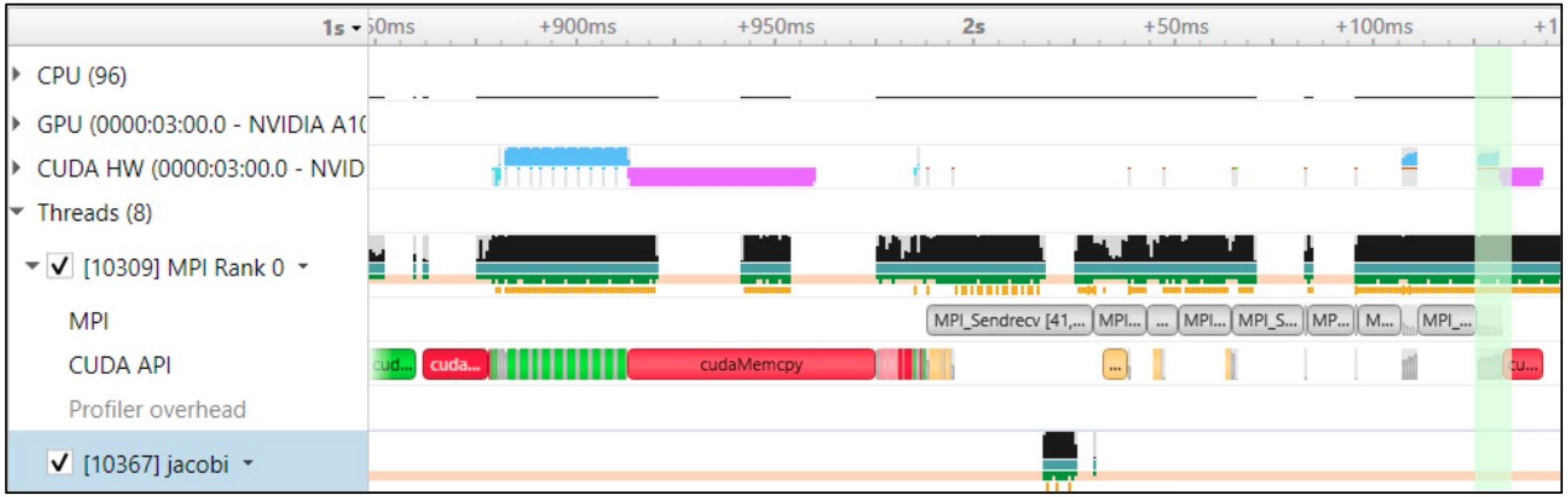
- Global timeline view
  - CUDA HW: streams, kernels, memory
- Different traces, e.g. CUDA, MPI
  - correlations API <-> HW
- Stack samples
  - bottom-up, top-down for CPU code
- GPU metrics
- Events View
  - Expert Systems
- looks at single process (tree)
  - correlate multi-process reports in single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

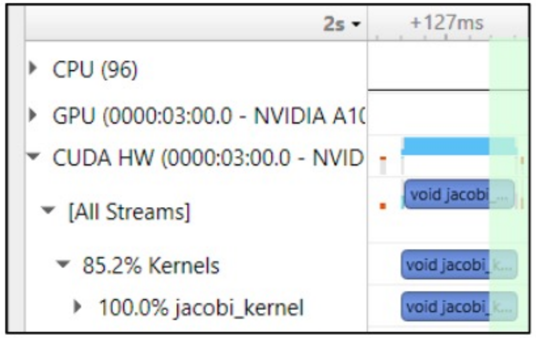


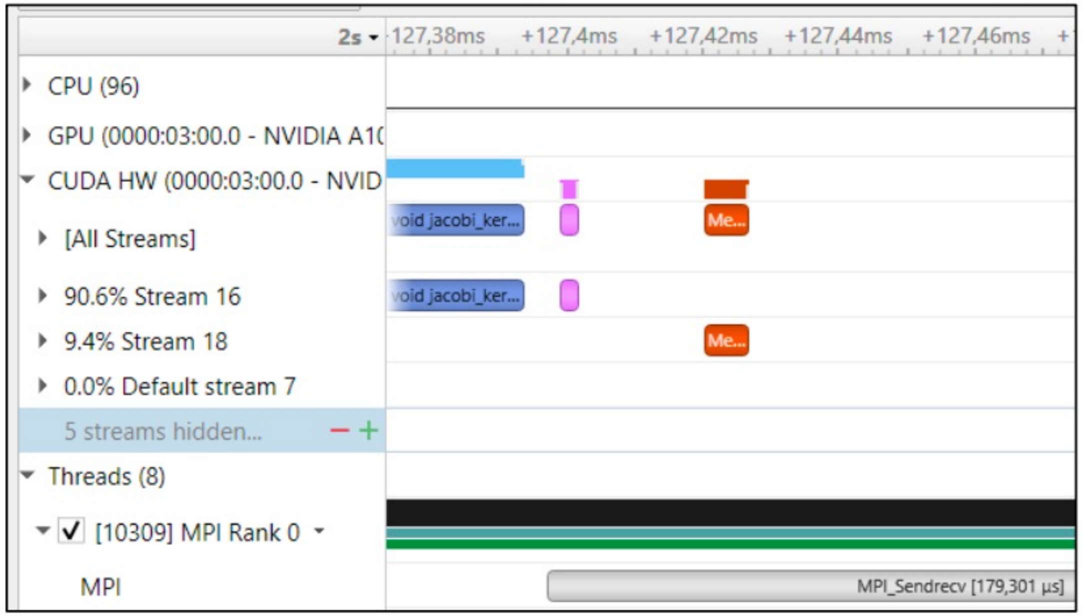
#	Name	Start	Duration	TID	GPU	Context
4	Memset	1,88258s	3,200 µs	-	GPU 0	Stream 13
5	void jacobi_ke...	1,88259s	3,056 ms	-	GPU 0	Stream 13
6	Memcpy DtoD	1,88565s	5,024 µs	-	GPU 0	Stream 14
7	Memcpy DtoH	1,88565s	4,864 µs	-	GPU 0	Stream 13

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
}
```

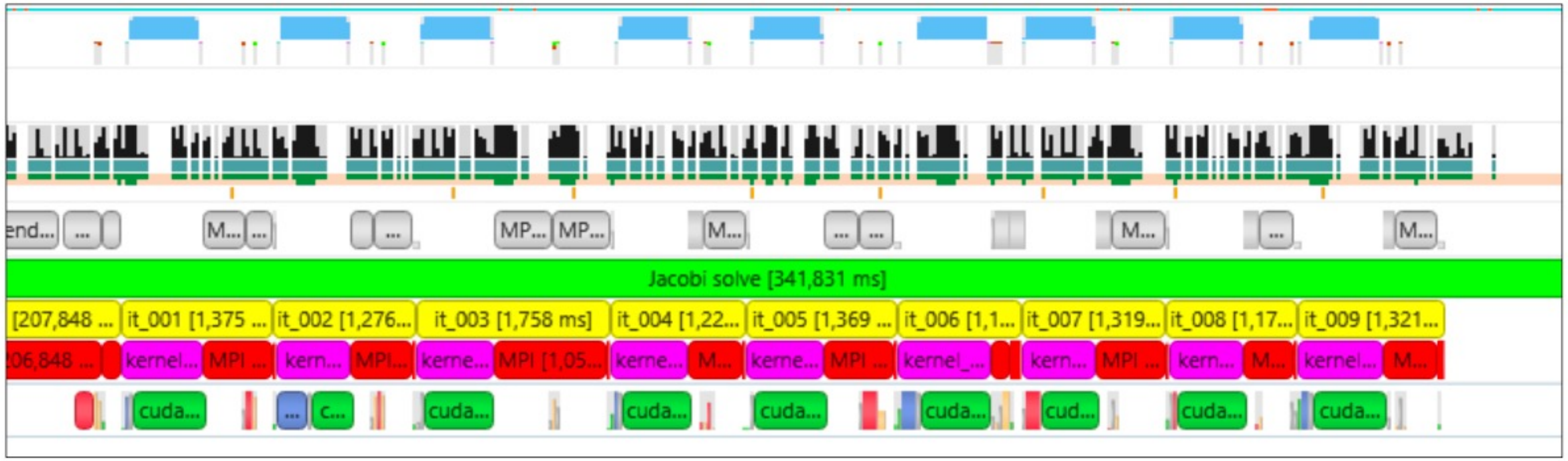
https://github.com/NVIDIA/NVTX and https://nvidia.github.io/NVTX/#how-do-i-use-nvtx-in-my-code

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

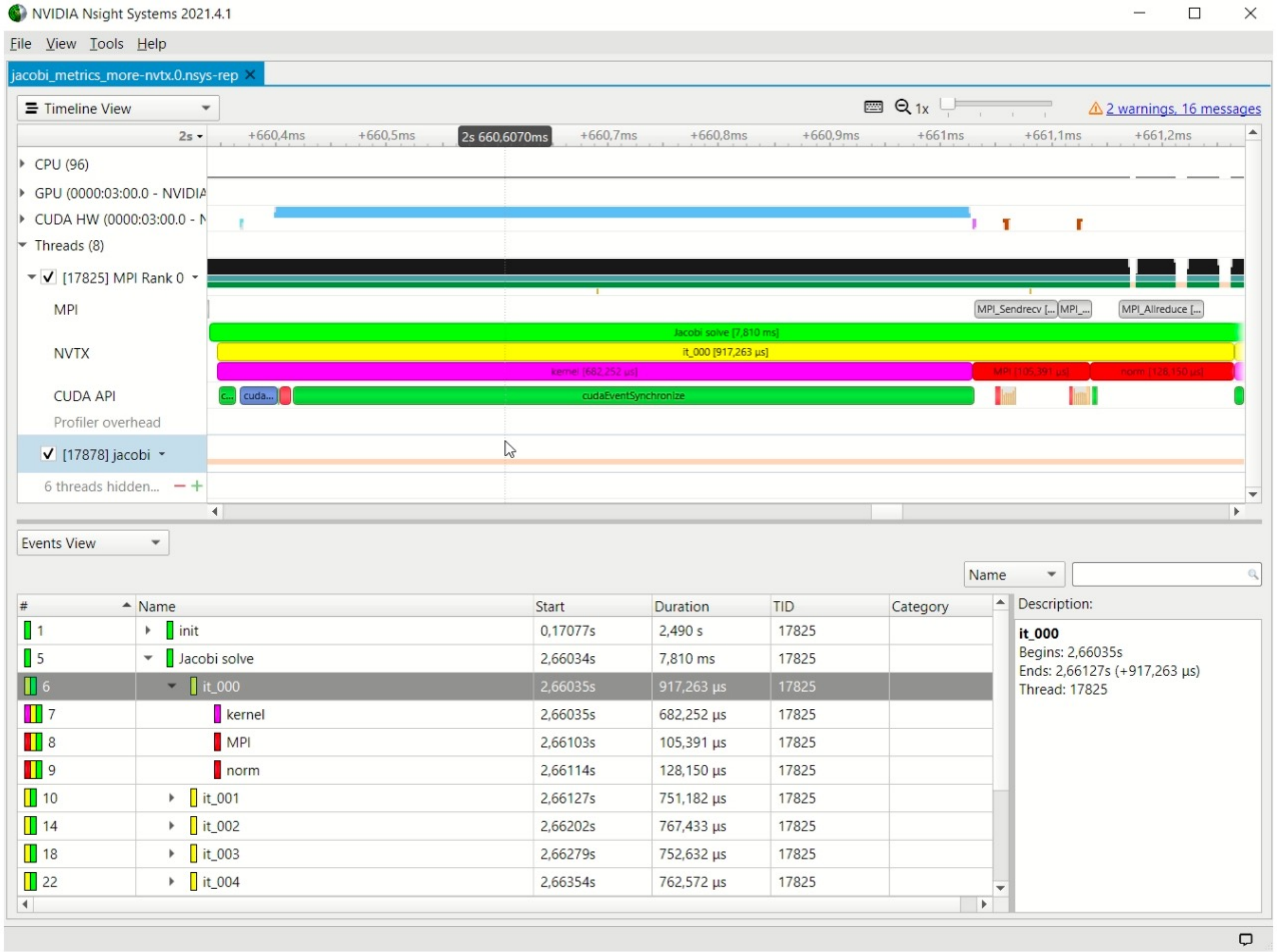
Shorter time, smaller files = quicker progress

- 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



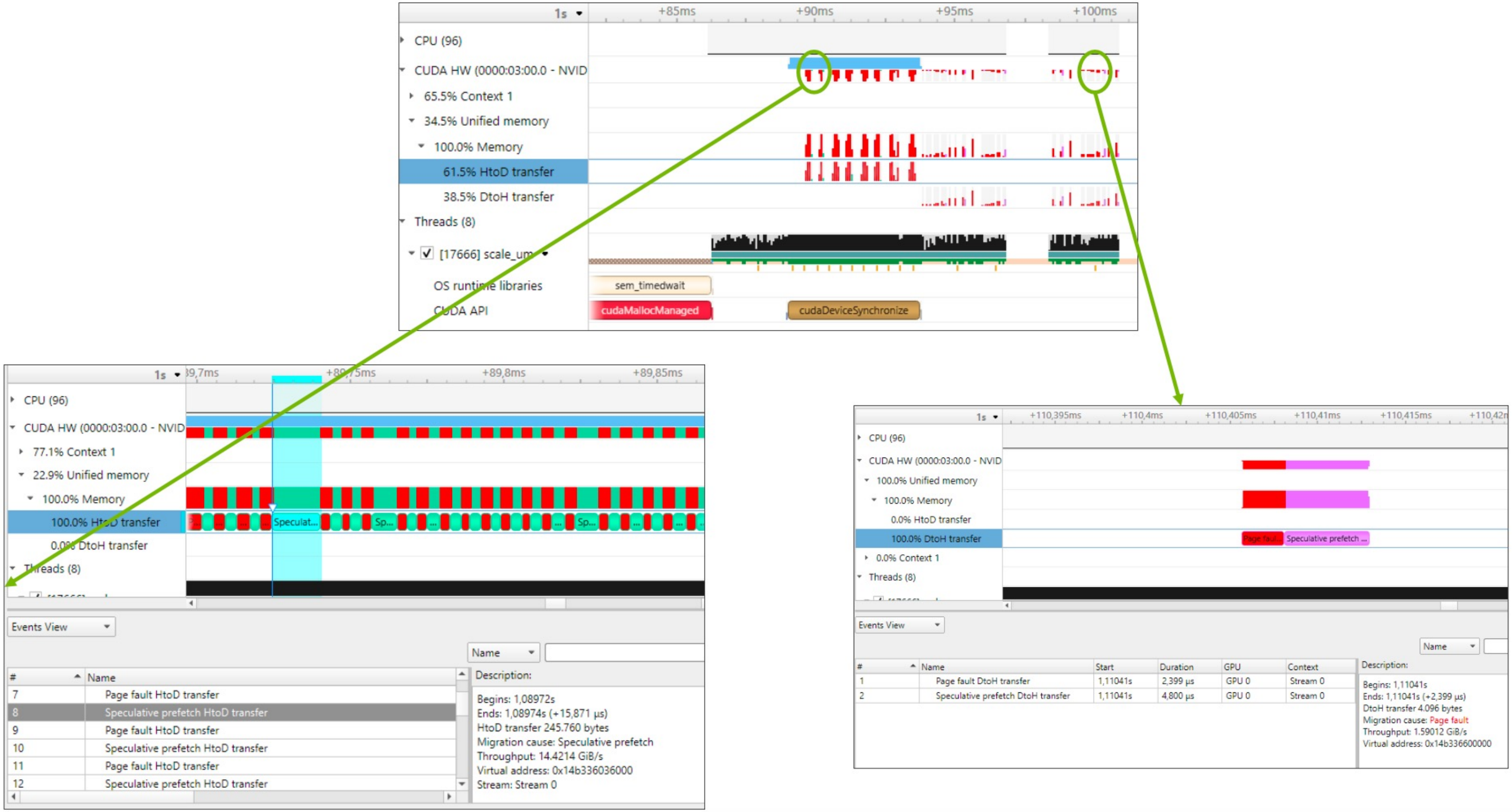
GPU Metrics in Nsight Systems

...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
```

Unified Memory movement

Observing transfers in Nsight Systems



THANK YOU

Instructor: Dr. Momme Allalen

www.nvidia.com/dli

```julia
using BenchmarkTools
using CUDA

using QXContexts

function main(args)
    file_path  = @__DIR__
    dsl_file   = joinpath(dirname(dirname(file_path)), "examples/ghz/ghz_5.qx")
    input_file = joinpath(dirname(dirname(file_path)), "examples/ghz/ghz_5.jld2")

    cg, _ = parse_dsl_files(dsl_file, input_file)

    # get time on gpu
    ctx_gpu = QXContext{CuArray{ComplexF32}}(cg)
    set_open_bonds!(ctx_gpu)
    # run to ensure all is precompiled
    t = NVTX.@range "Warm up" begin @elapsed ctx_gpu() end
    @info "GPU warmup ran in $t"
    CUDA.@profile NVTX.@range "Run iteration" begin
        ctx_gpu()
    end
    nothing
end

main(ARGS)
```

Demo using NVTX and Nsight