NVIDIA

PROFILING OF GPU PROGRAMS

Markus Hrywniak, DevTech Compute, 22 January 2021

OVERVIEW

No prior GPU (profiling) experience required here

- Examples use GPU, but no requirement for method

Variety of different tools, possibilities for profiling

- Recommend: Start with Nsight Systems
- Sampling, system-wide, CUDA traces (+ more)

Many people do not use profilers – perceived difficulty?

- Most important lesson: „Do not trust your gut“
- Let‘s start with profiling basics!

WHAT DOES A PROFILER DO?

Sampling vs. Instrumentation (very simplified)

Every ms, take a sample of callstack

```
while (...) {
  do_nothing()
  intense_calculation()
  sleep()
}
```

Samples
0
23
12

(+) Hot spots show up, low overhead

(-) May miss some calls

Instrument function calls, APIs, etc. (automatable)

```
while (...) {
  trace_do_nothing() -> do_nothing()
  trace_intense_calculation() ->   intense_calculation()
  trace_sleep() ->  sleep()
}
```

(+) Captures whole program, full call chains

(-) Potentially higher overhead, skew

THE NSIGHT SUITE

Application-wide profiling (Systems), Kernel-level profiling (Compute)

Tracing: CUDA API calls, MPI trace, OpenACC trace, OpenMP

Sampling, hardware counters

Augment with NVIDIA Tools Extension (NVTX):
Create (nested) ranges, define macros

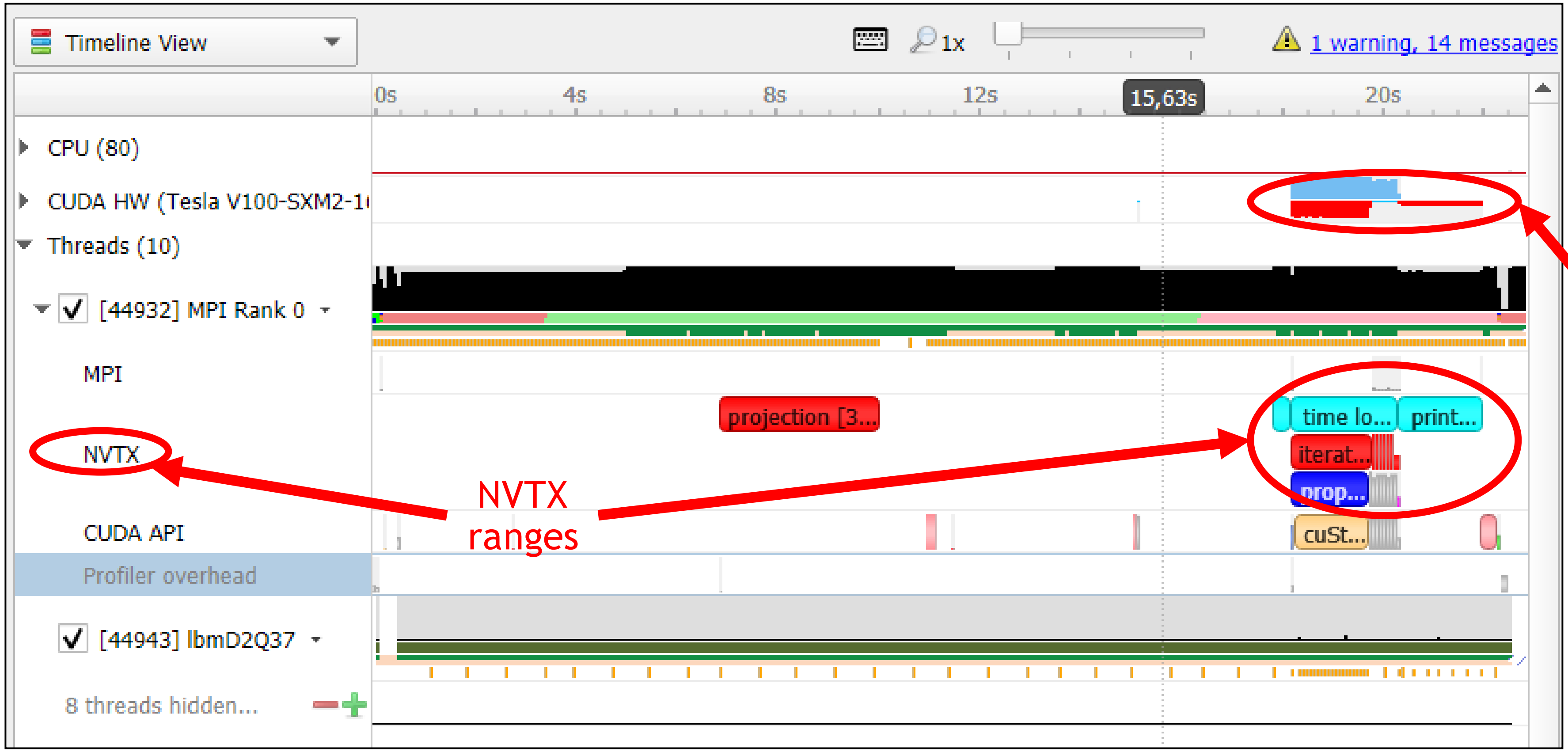
Nsight Systems

Nsight Compute

NVTX primer: https://devblogs.nvidia.com/parallelforall/cuda-pro-tip-generate-custom-application-profile-timelines-nvtx/

WHETTING YOUR APPETITE

Timeline overview in Nsight Systems GUI



Here: Application already ported to GPU – basic guidelines followed (coalescing, data movement, SoA)

S7122: CUDA Optimization Tips, Tricks and Techniques (2017)

A FIRST (I)NSIGHT

Maximum achievable speedup: Amdahl‘s law

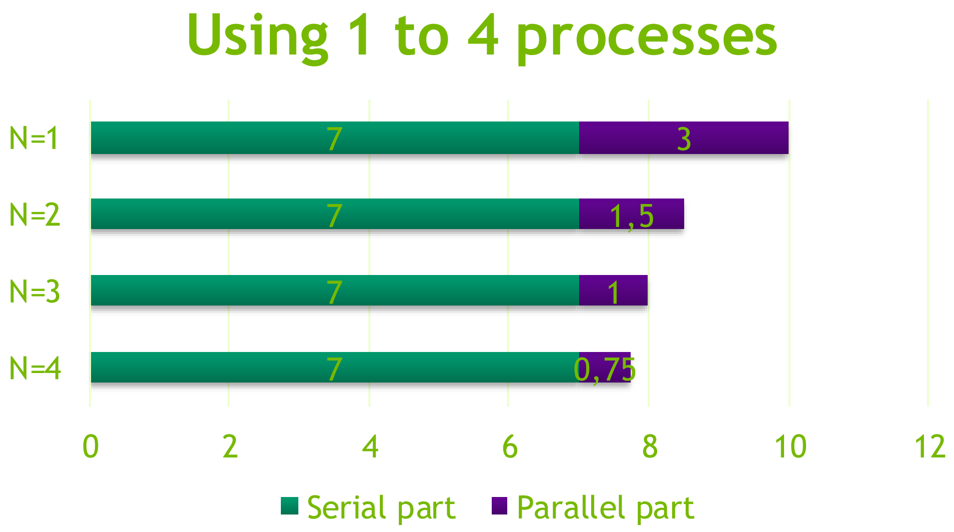
Amdahl‘s law states overall speedup $\boldsymbol{s}$ given the parallel fraction p of code and number of processes N

$$\boldsymbol{s} = \frac{1}{1 - p + \frac{p}{N}} < \frac{1}{1 - p}$$

Limited by serial fraction, even for $N \to \infty$

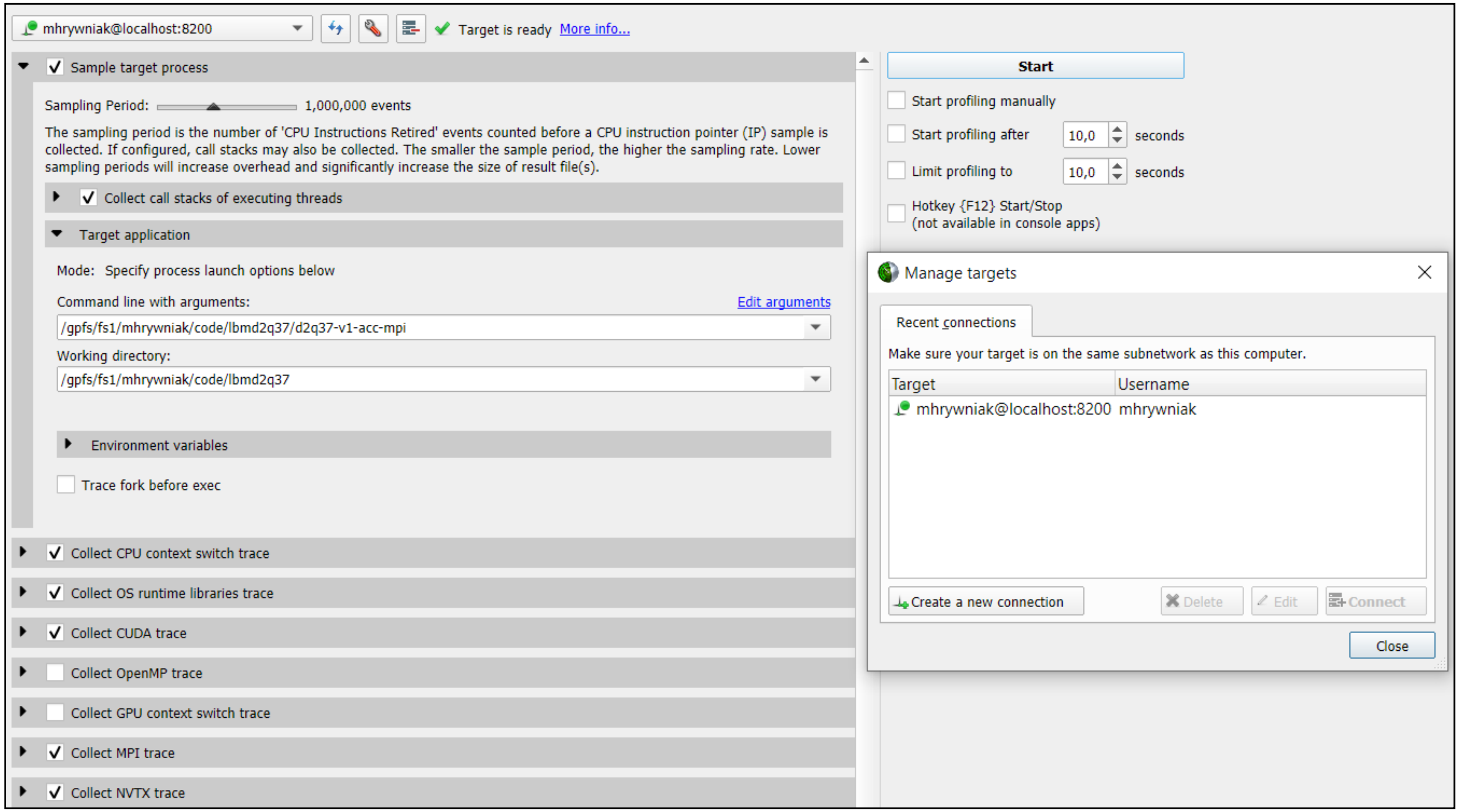
Example for $p = 30\%$

Also valid for per-method speedups



A FIRST (I)NSIGHT

Recording with the GUI



Connect directly

Or use an SSH Tunnel:
`ssh -L 8200:`*`compute-node`*`:22`*` login-node`*

Select traces to collect

A FIRST (I)NSIGHT

Recording an application timeline

1) We‘ll use the command line

```
mpirun –np $NP \
nsys profile --trace=cuda,nvtx,mpi \
--output=my_report.%q{OMPI_COMM_WORLD_RANK}.qdrep ./myApp
```

Note: Slurm users, try `srun ... %q{SLURM_PROCID}`

2) Inspect results: Open the report file in the GUI

Also possible to get details on command line (documentation), `nsys stats --help`

See also https://docs.nvidia.com/nsight-systems/, "Profiling from the CLI on Linux Devices"

THE GENERAL IDEA

Application timeline with Nsight Systems

Create reduced test case that hits important code paths

Profile once and look at structure

Augment & Annotate analyzed regions (NVTX)

Identify optimization targets (Wallclock, Amdahl‘s law)

Fill „blank spots“ on timeline with GPU activity

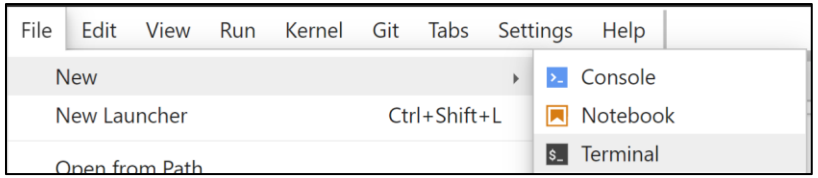
Now: Let‘s see how to start using this on your codes and on JUWELS

1) Simple example

2) Real-world application

ON JUWELS

Recording a profile from JupyterLab



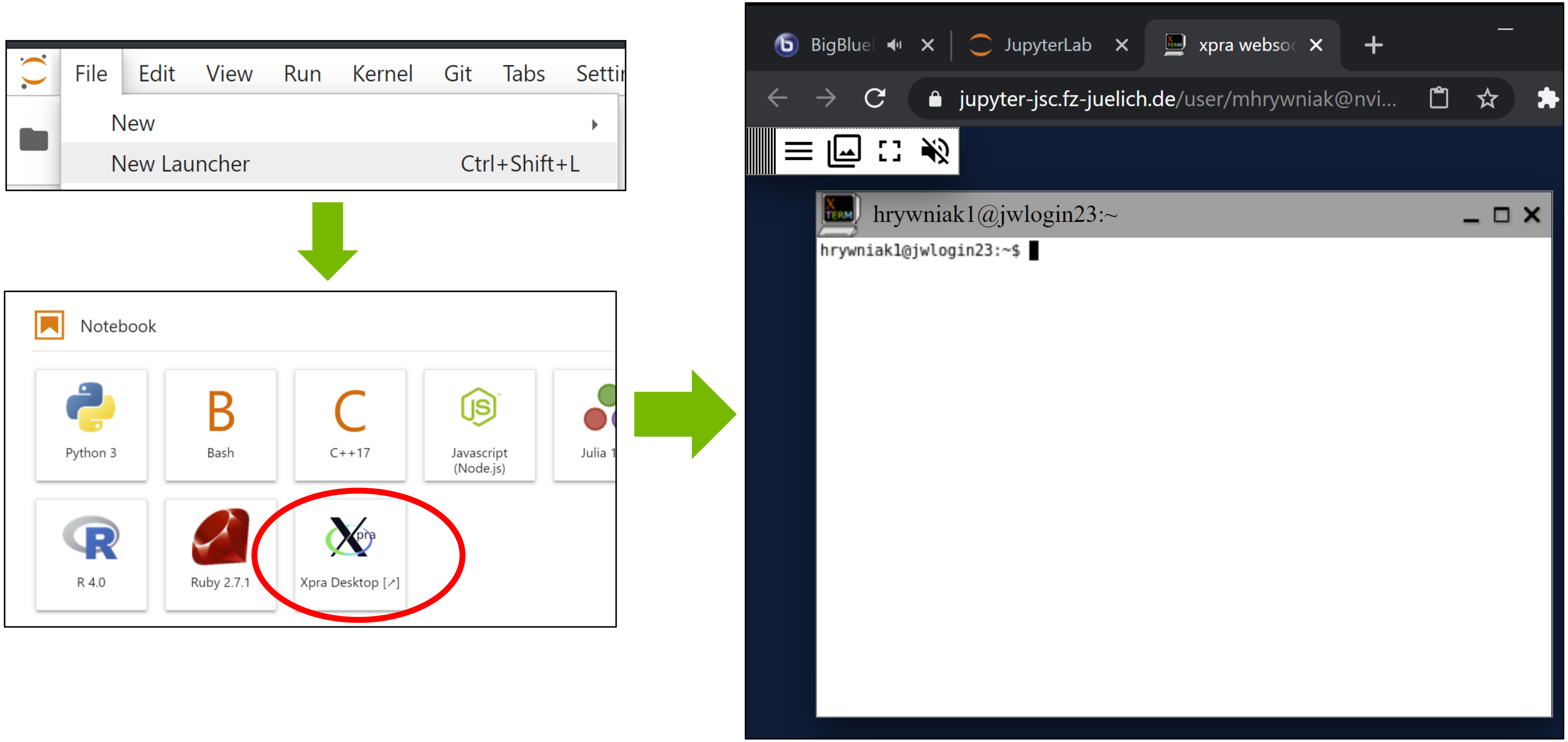
1. Terminal
2. Load modules, launch profiling command
   1. `module load CUDA GCC Nsight-Systems # see below for versions`
   2. `srun -n1 -N1 nsys profile -f true -o my_report ./executable`
   3. Generates and overwrites my_report.qdrep
3. Open the report in `nsys-ui` or use `nsys stats my_report.qdrep`

Caveat: Load `Nsight-Systems/2020.4.1` since 2020.5.1 has a bug with JUWELS‘ srun (fixed in next version)

File naming for MPI with `my_report.%q{SLURM_PROCID}` (see MPI talk)

LAUNCHING THE GUI

Xpra X-forwarding in your browser



LAUNCHING THE GUI

Xpra X-forwarding in your browser

Caveat: You need to re-load your modules - this is a separate session

```
module load Nsight-Systems
nsys-ui
```

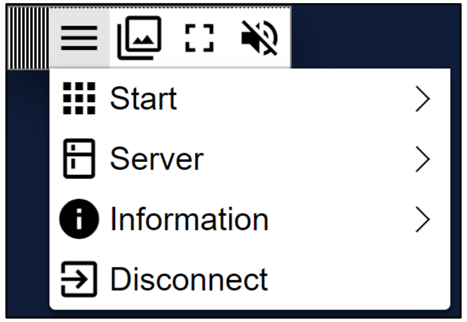
Newer nsys versions can open older reports

Xpra: Note the bar, in case you close your terminal...

Generally works very well (zero setup)

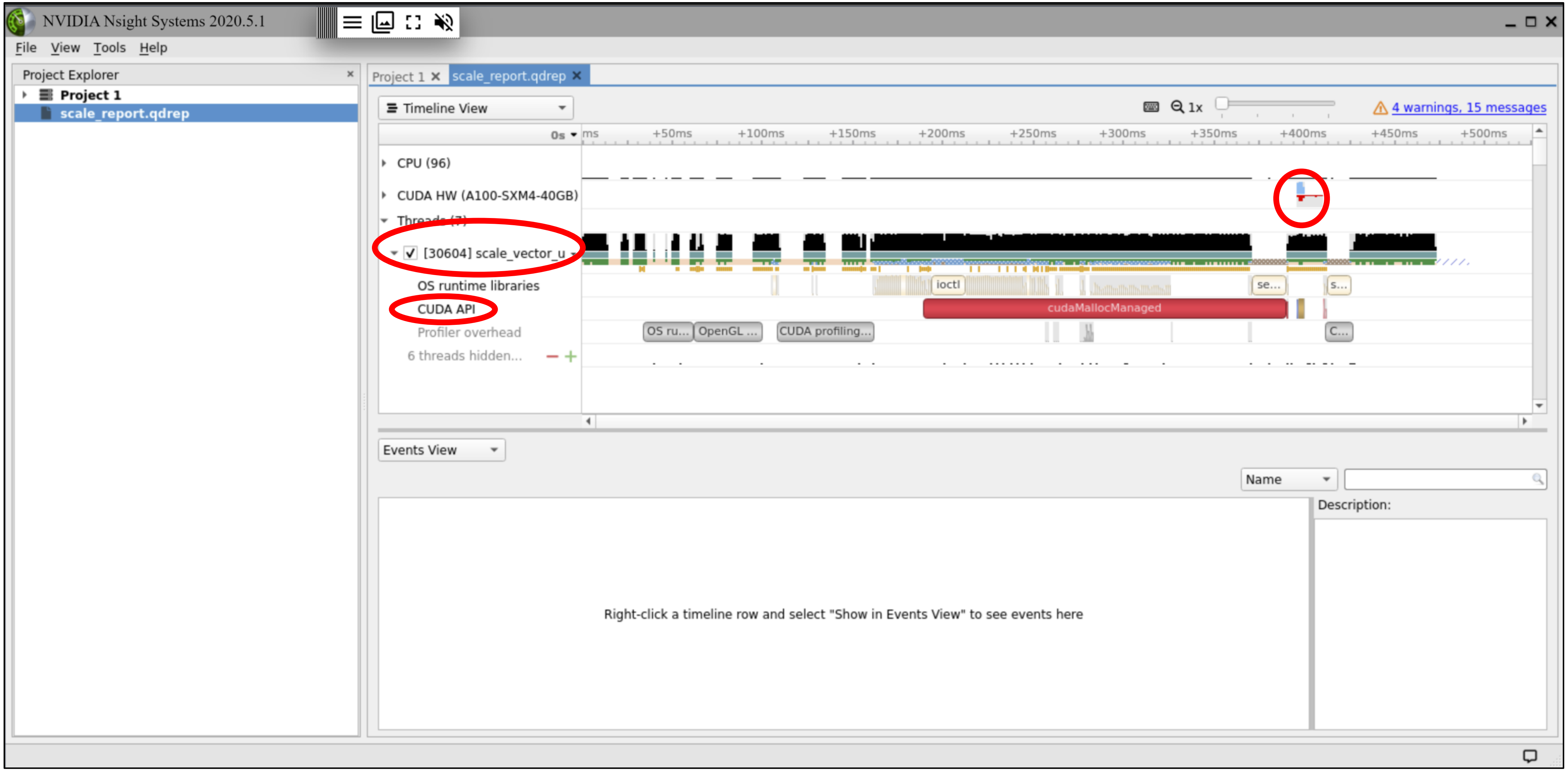
For heavy usage: Nsight Systems is free, can install it locally

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

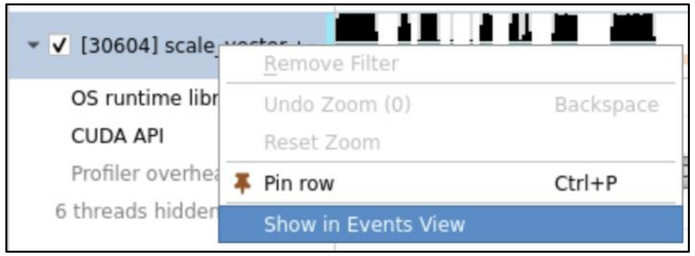


NVIDIA

LOOKING AT A SIMPLE EXAMPLE



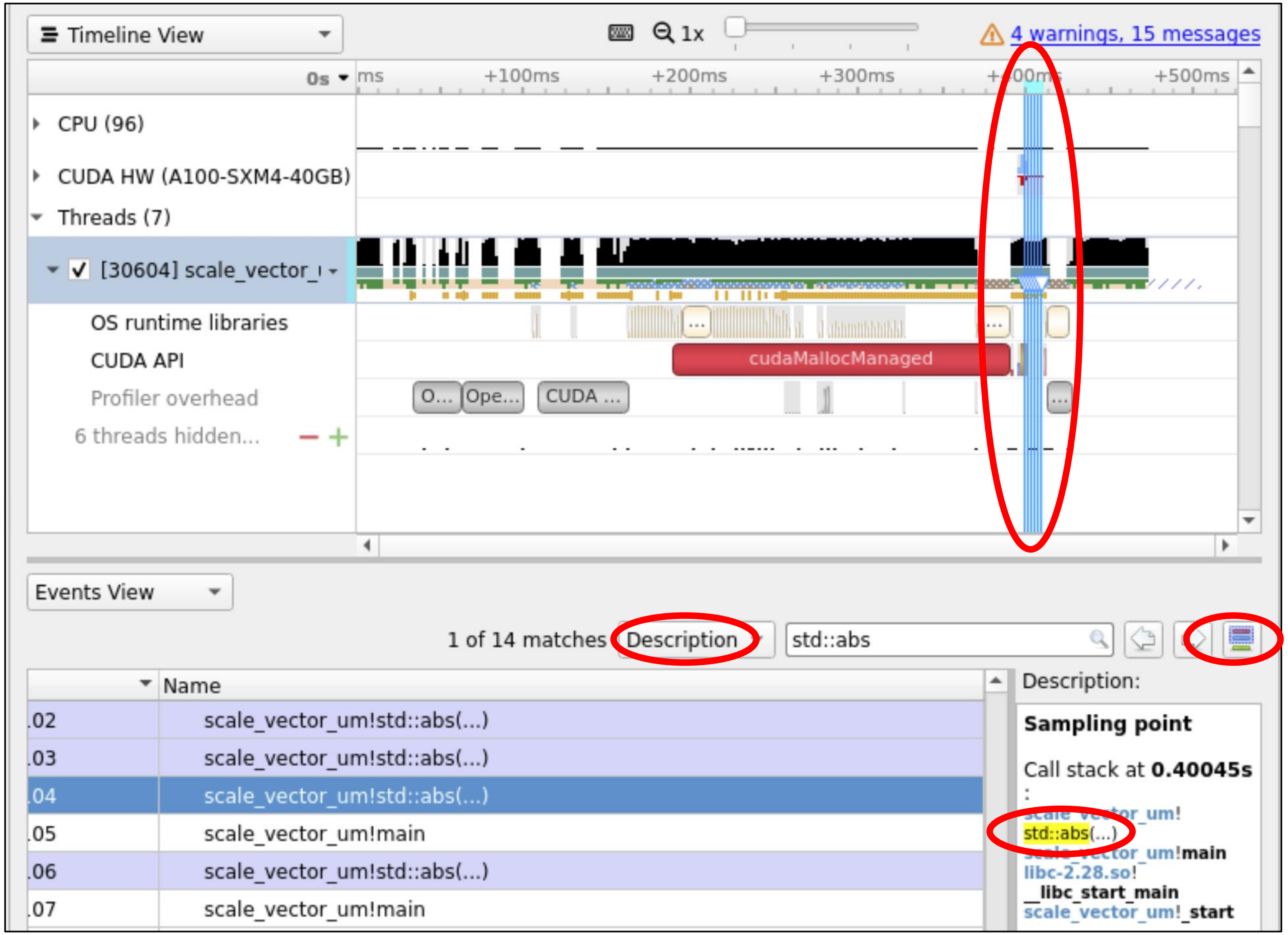
USING CALLSTACK SAMPLES



Events View makes information searchable

„Highlight All“ shows all matches

Can search in description, includes callstack



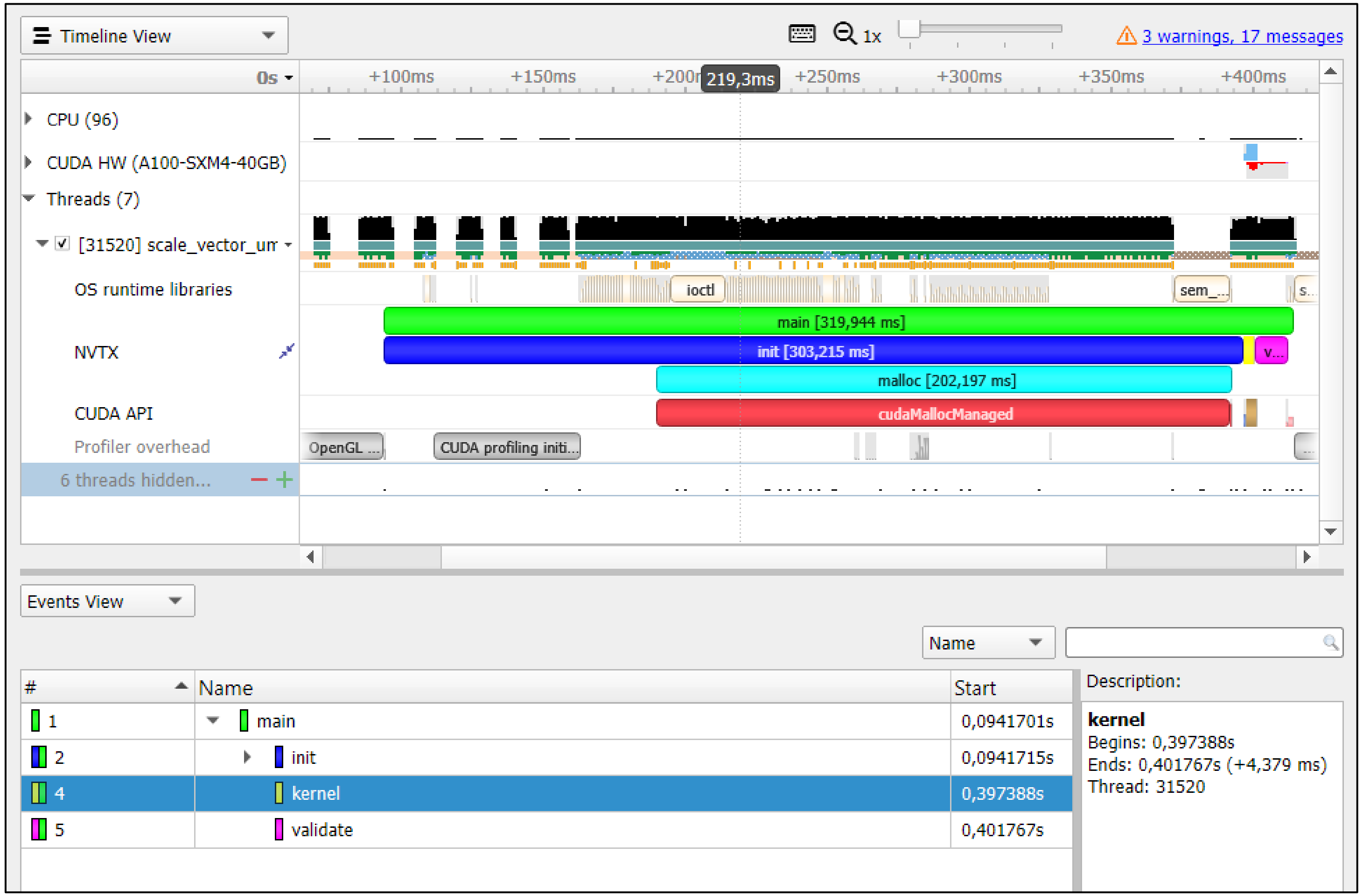
ADDING SOME COLOR

Code annotation with NVTX

Like manual timing, only less work

Nesting, timing

Correlation, filtering



ADDING NVTX

Simple range-based API

```
#include <nvToolsExt.h>
```

Copy&paste PUSH/POP macros (or module)

```
        PUSH(name, color)
```

Sprinkle them strategically through code

NVTX v3 is header-only

Not shown: Advanced usage (domains, ...)

https://github.com/NVIDIA/NVTX

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

https://developer.nvidia.com/blog/customize-cuda-fortran-profiling-nvtx/

```
int main(int argc, char** argv){
    PUSH("main", 0)
    PUSH("init", 1)
```

```
    POP
    PUSH("kernel", 2)

    scale<<<gridDim, blockDim>>>(alpha, a, c, m);

    cudaDeviceSynchronize();
    POP
    PUSH("validate", 3)
```

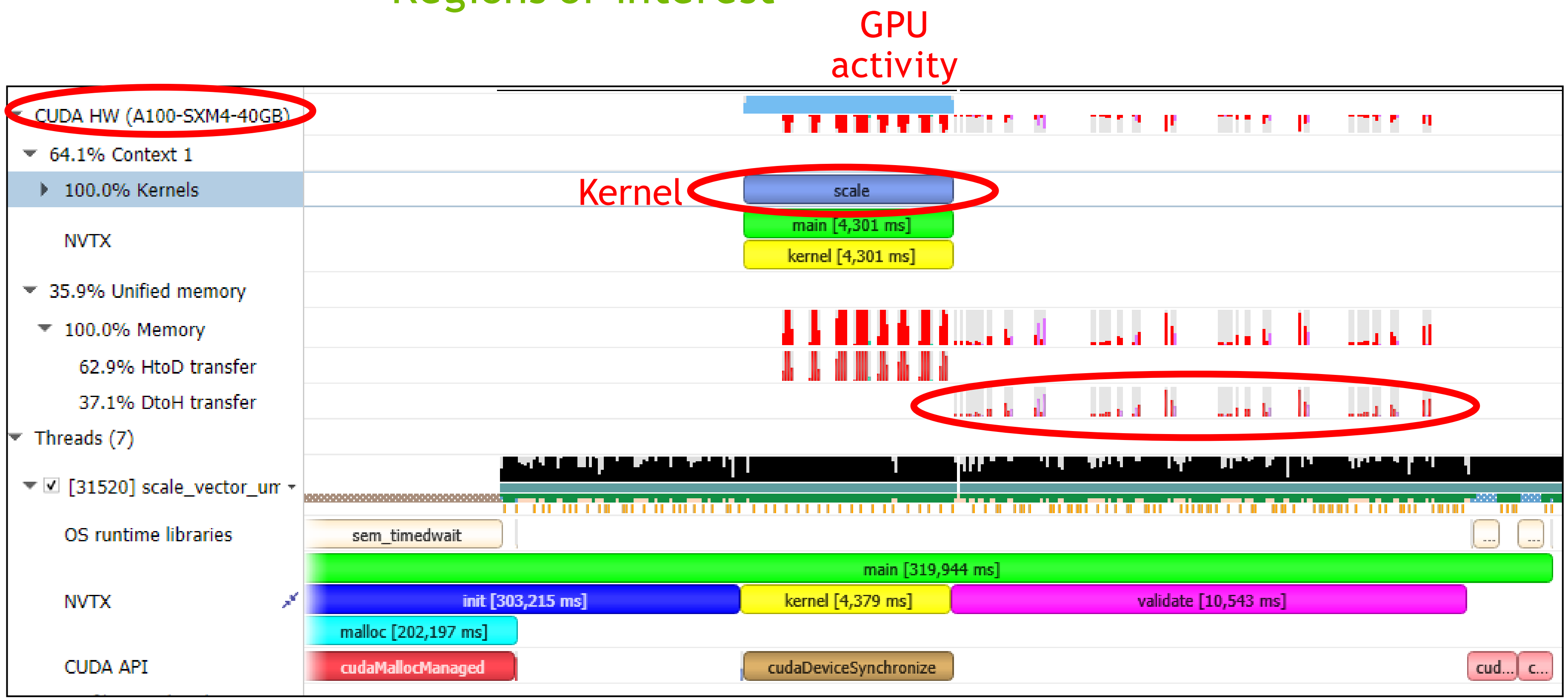
ZOOMING IN

Regions of interest

Kernel launch

UM migrations and page faults

Use Amdahl's law as heuristic



MINIMIZING PROFILE SIZE

Shorter time, smaller files = quicker progress

Only profile what you need – all profilers have some overhead

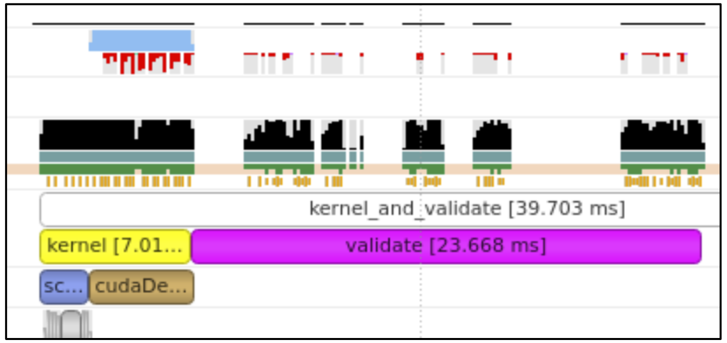
Bonus: lower number of events => smaller file size

Add to nsys command line:

```
--capture-range=nvtx --nvtx-capture=any_nvtx_marker_name \
--env-var=NSYS_NVTX_PROFILER_REGISTER_ONLY=0 --kill none
```

Alternatively: `cudaProfilerStart()` and `–Stop()`

```
--capture-range=cudaProfilerApi
```



NVIDIA

OTHER FEATURES

We only covered a small subset

„Traditional“ top-down or bottom-up stack views

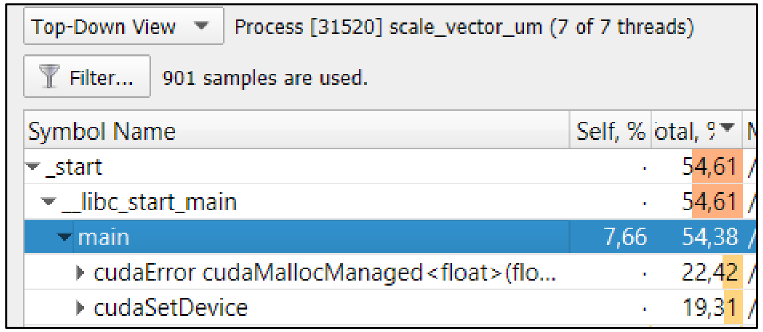
Lots of different traces (MPI, OpenACC, OpenMP, ...)

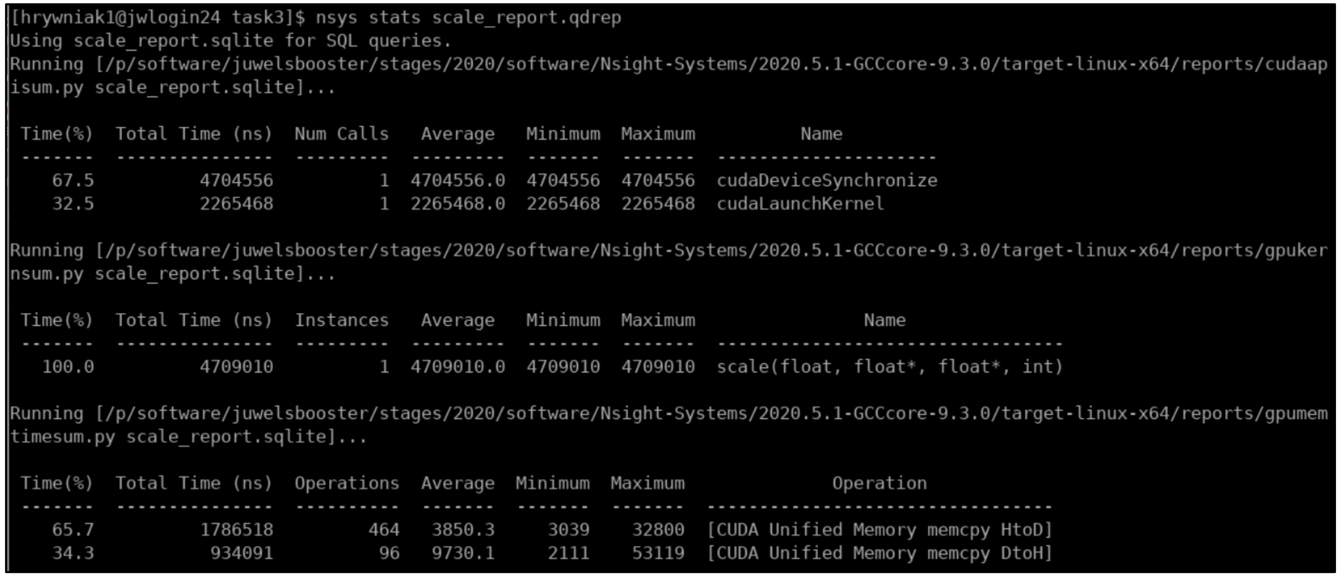
Data export (csv, sqlite, ...)

Customizable reports via Python scripts

Full guide:

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





```
[hrywniak1@jwlogin24 task3]$ nsys stats scale_report.qdrep
Using scale_report.sqlite for SQL queries.
Running [/p/software/juwelsbooster/stages/2020/software/Nsight-Systems/2020.5.1-GCCcore-9.3.0/target-linux-x64/reports/cudaap
isum.py scale_report.sqlite]...

 Time(%)  Total Time (ns)  Num Calls   Average   Minimum  Maximum          Name
 -------  ---------------  ---------  ---------  -------  -------  --------------------
    67.5          4704556          1  4704556.0  4704556  4704556  cudaDeviceSynchronize
    32.5          2265468          1  2265468.0  2265468  2265468  cudaLaunchKernel

Running [/p/software/juwelsbooster/stages/2020/software/Nsight-Systems/2020.5.1-GCCcore-9.3.0/target-linux-x64/reports/gpuker
nsum.py scale_report.sqlite]...

 Time(%)  Total Time (ns)  Instances   Average   Minimum  Maximum                Name
 -------  ---------------  ---------  ---------  -------  -------  ----------------------------------
   100.0          4709010          1  4709010.0  4709010  4709010  scale(float, float*, float*, int)

Running [/p/software/juwelsbooster/stages/2020/software/Nsight-Systems/2020.5.1-GCCcore-9.3.0/target-linux-x64/reports/gpumem
timesum.py scale_report.sqlite]...

 Time(%)  Total Time (ns)  Operations  Average  Minimum  Maximum              Operation
 -------  ---------------  ----------  -------  -------  -------  ---------------------------------
    65.7          1786518         464   3850.3     3039    32800  [CUDA Unified Memory memcpy HtoD]
    34.3           934091          96   9730.1     2111    53119  [CUDA Unified Memory memcpy DtoH]
```

WHEN TO MOVE ON

Proper tool for the job

Specialized MPI profiling/bottlenecks, load imbalance

Kernel-level profiling -> Nsight Compute

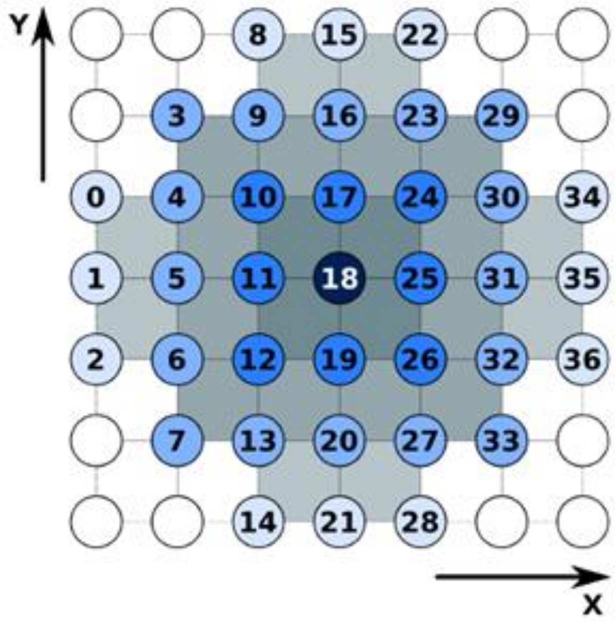
- Used later on (get the low-hanging fruit first!)
- Use it when you find a hotspot kernel

Now: Revisit the real-world example from the beginning

KNOW YOUR CODE

Overview of LBM D2Q37* algorithm phases

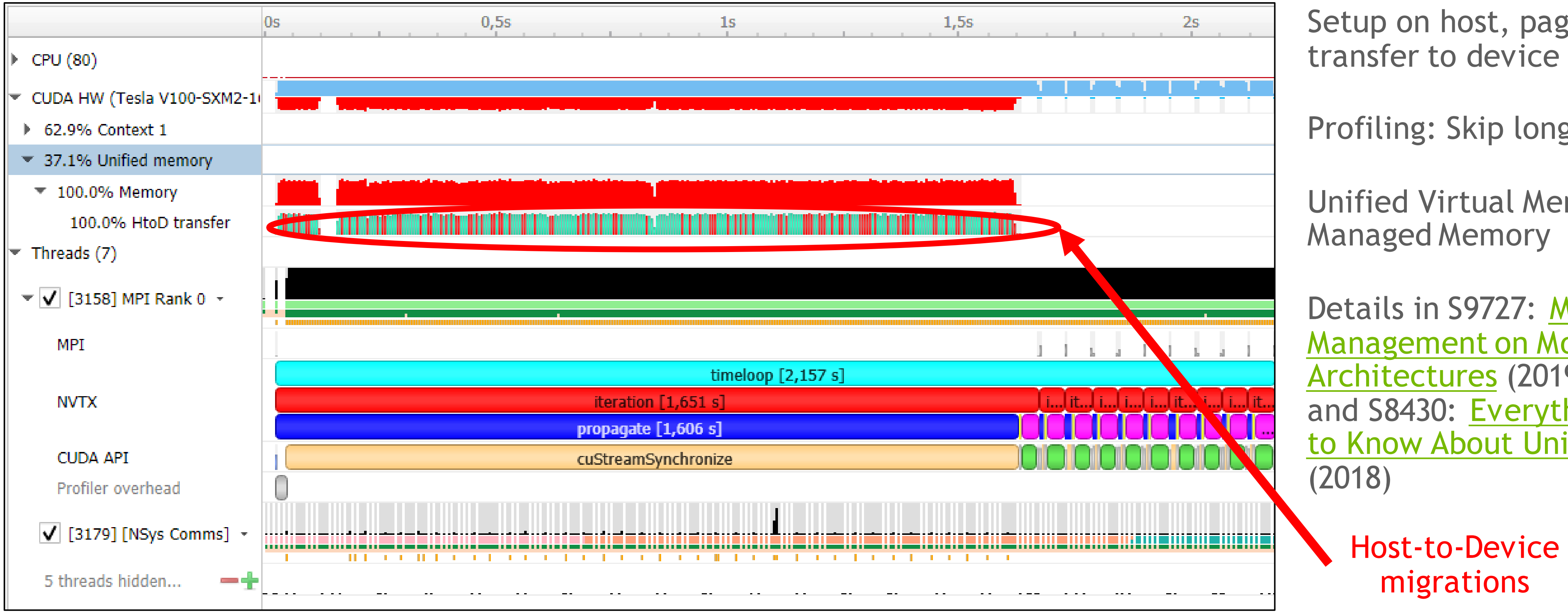
1. One-time setup: Create and fill data structures
2. Time loop
    1. Propagate
    2. Boundary conditions
    3. Collide
3. Output and finalization



*Details on code and in-depth analysis: What the Profiler Is Telling You (GTC 2020)

LOOKING CLOSER

Focusing on the time loop



Setup on host, page fault, transfer to device

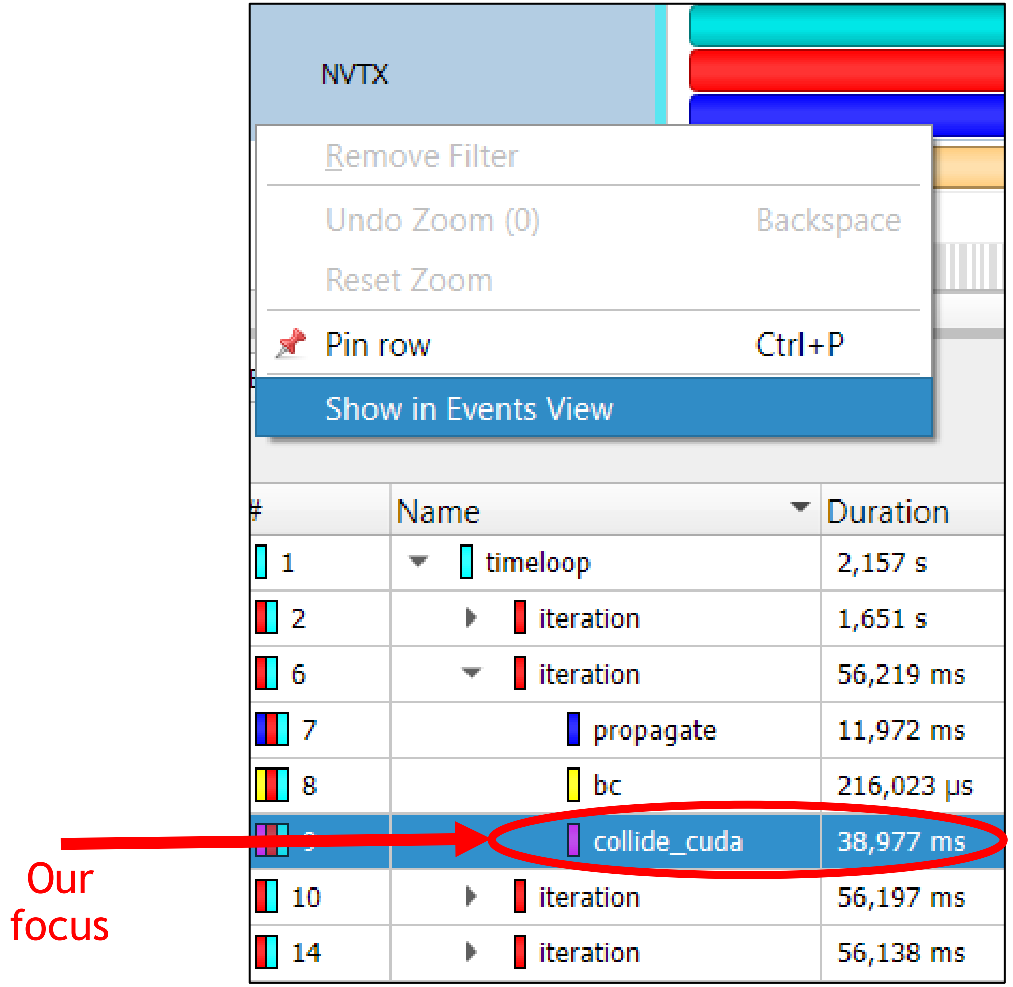
Profiling: Skip long first iteration

Unified Virtual Memory, Managed Memory

Details in S9727: Memory Management on Modern GPU Architectures (2019) and S8430: Everything You Need to Know About Unified Memory (2018)

LOOKING CLOSER

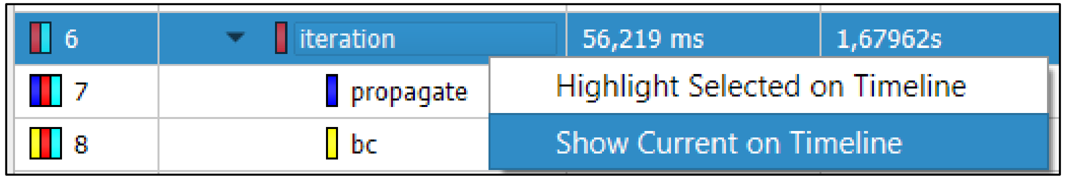
Focusing on the iteration



Zooming in and using Events View for NVTX

Useful for other rows, e.g. CUDA API

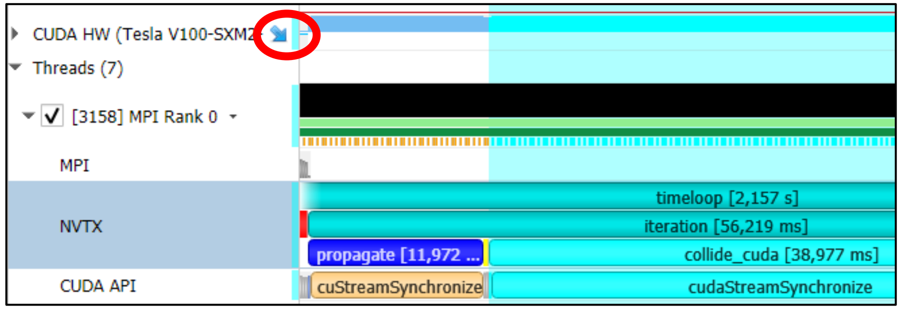
Hierarchy of ranges, use to locate on timeline:



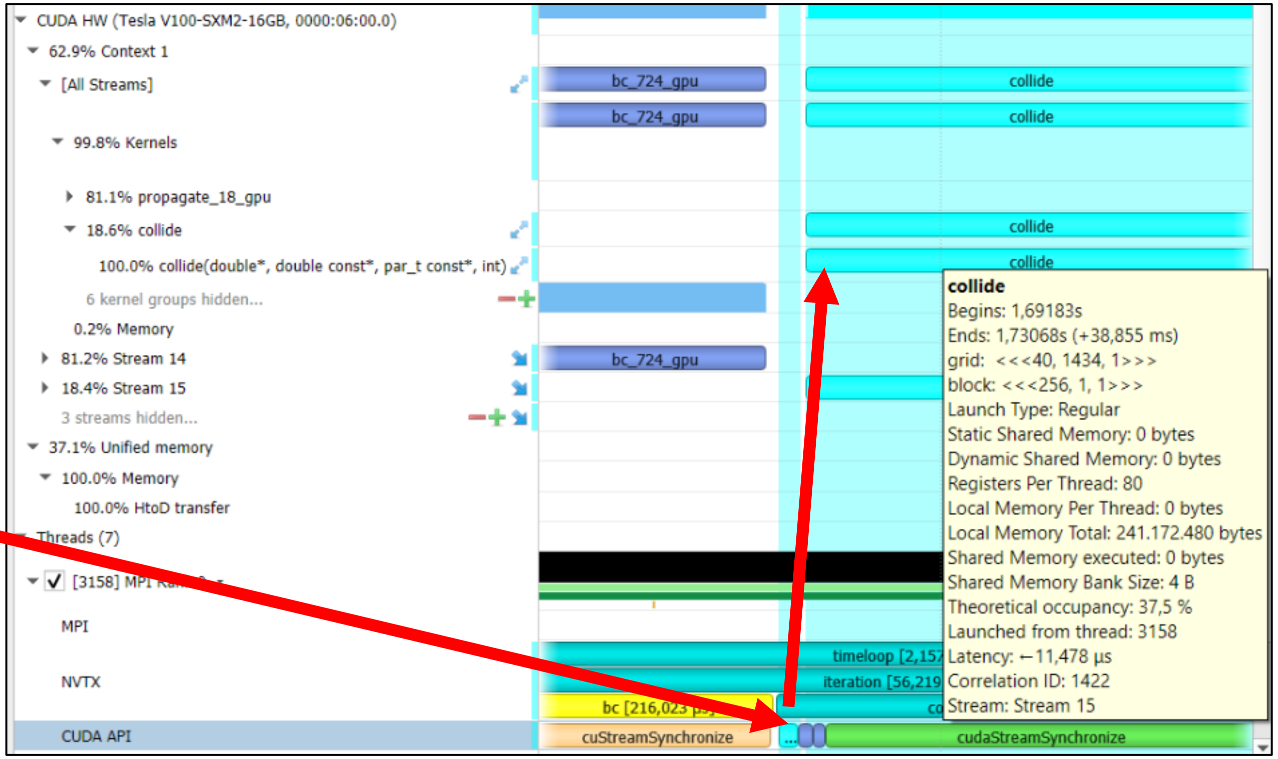
IDENTIFYING INTERESTING REGIONS

How to correlate ranges, API and kernel calls

Basic block NVTX „iteration“. Identify components. Mark kernel in CUDA API row, find kernel launch



Finding correlations



SUMMARY

How to approach porting your own code

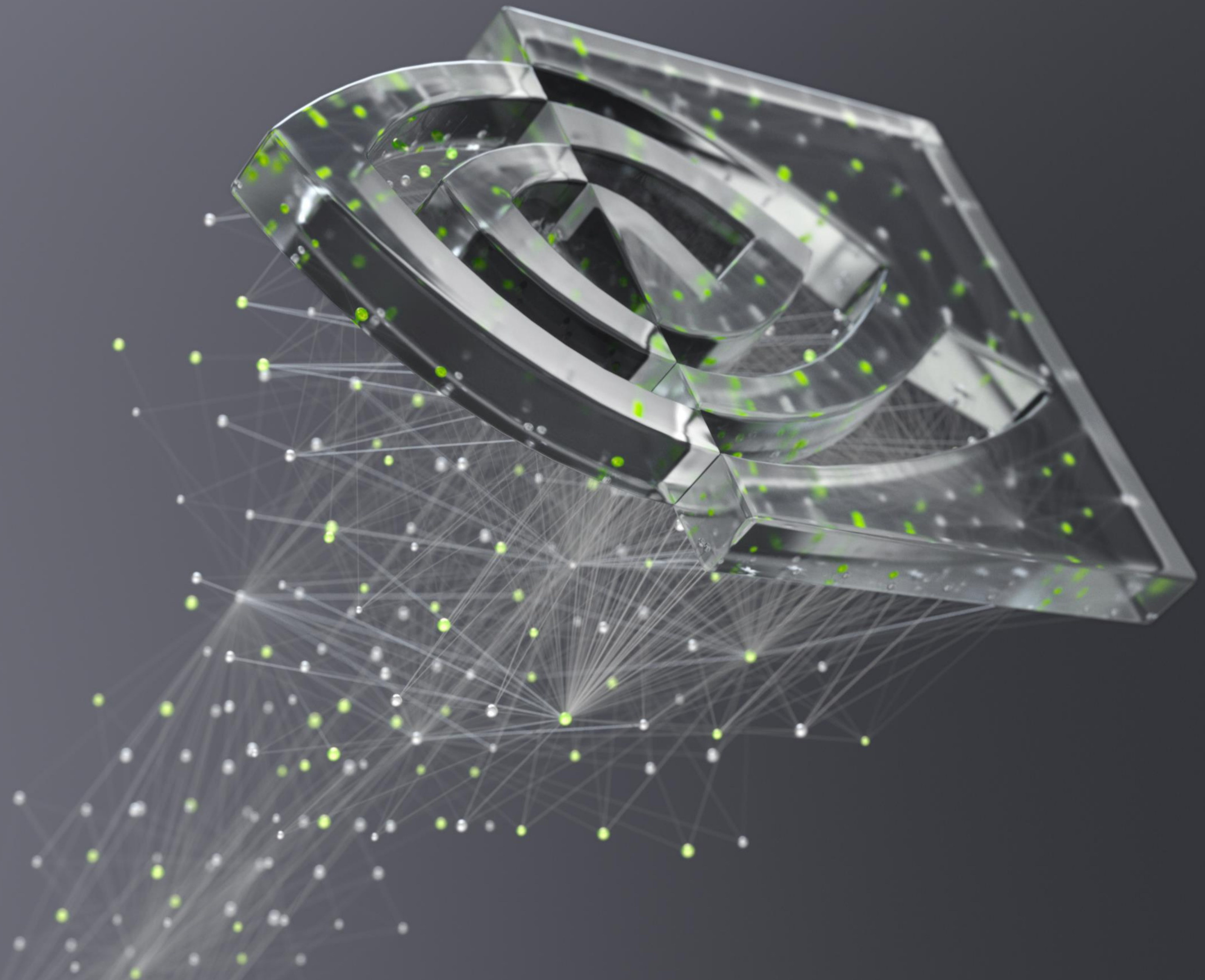
Start with Nsight Systems and record a first profile

Identify roughly some features (use call stacks, code knowledge), add NVTX

- Add and customize traces as needed
- Use capture ranges

Iteratively eliminate „blank“ spots – is the GPU active?

Switch to more specialized profilers as needed



NVIDIA