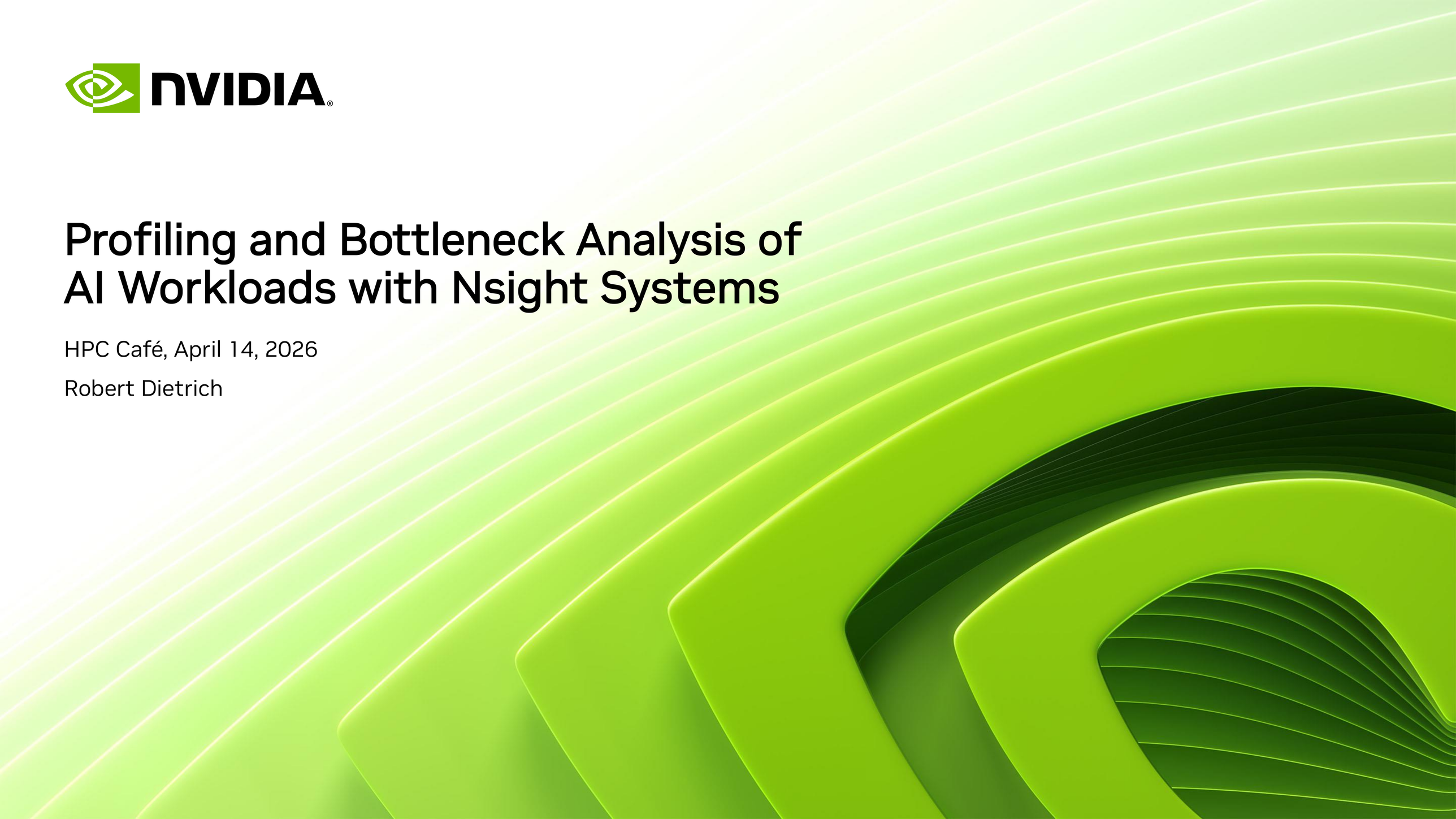


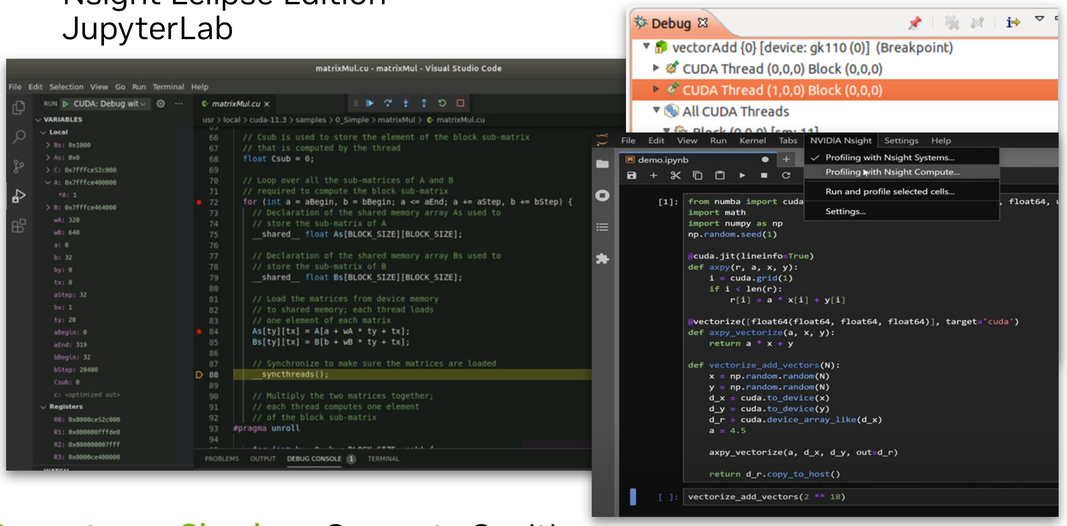
# Profiling and Bottleneck Analysis of AI Workloads with Nsight Systems

HPC Café, April 14, 2026

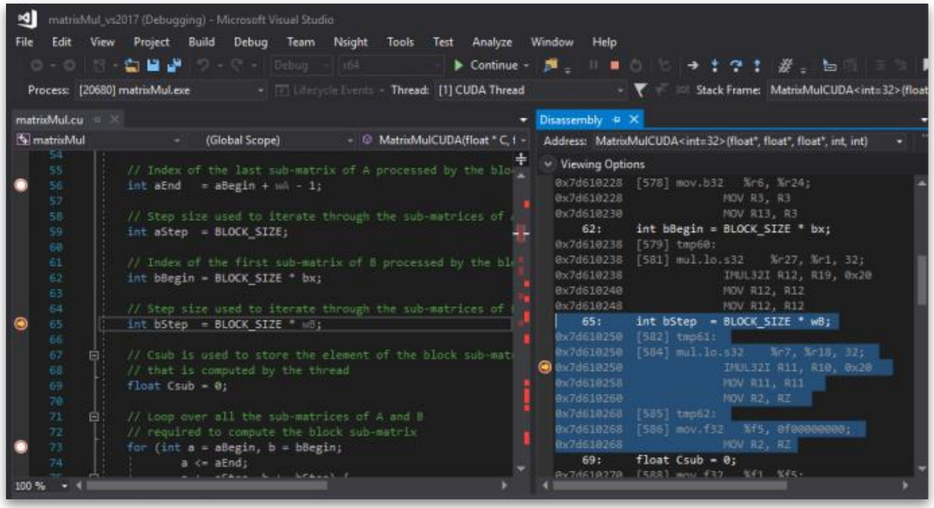
Robert Dietrich

# Developer Tools Ecosystem

**IDE integrations**: Nsight Visual Studio **Code** Edition
Nsight Visual Studio Edition
Nsight Eclipse Edition
JupyterLab



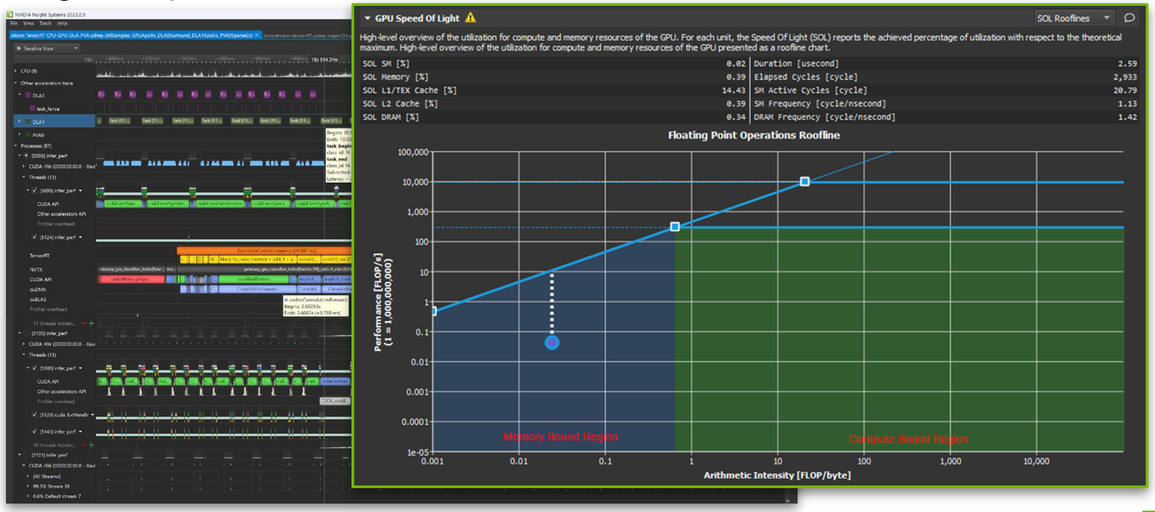

**Debuggers**: CUDA-GDB, Nsight Visual Studio Edition
Nsight Visual Studio **Code** Edition, Nsight Compute, Nsight Graphics



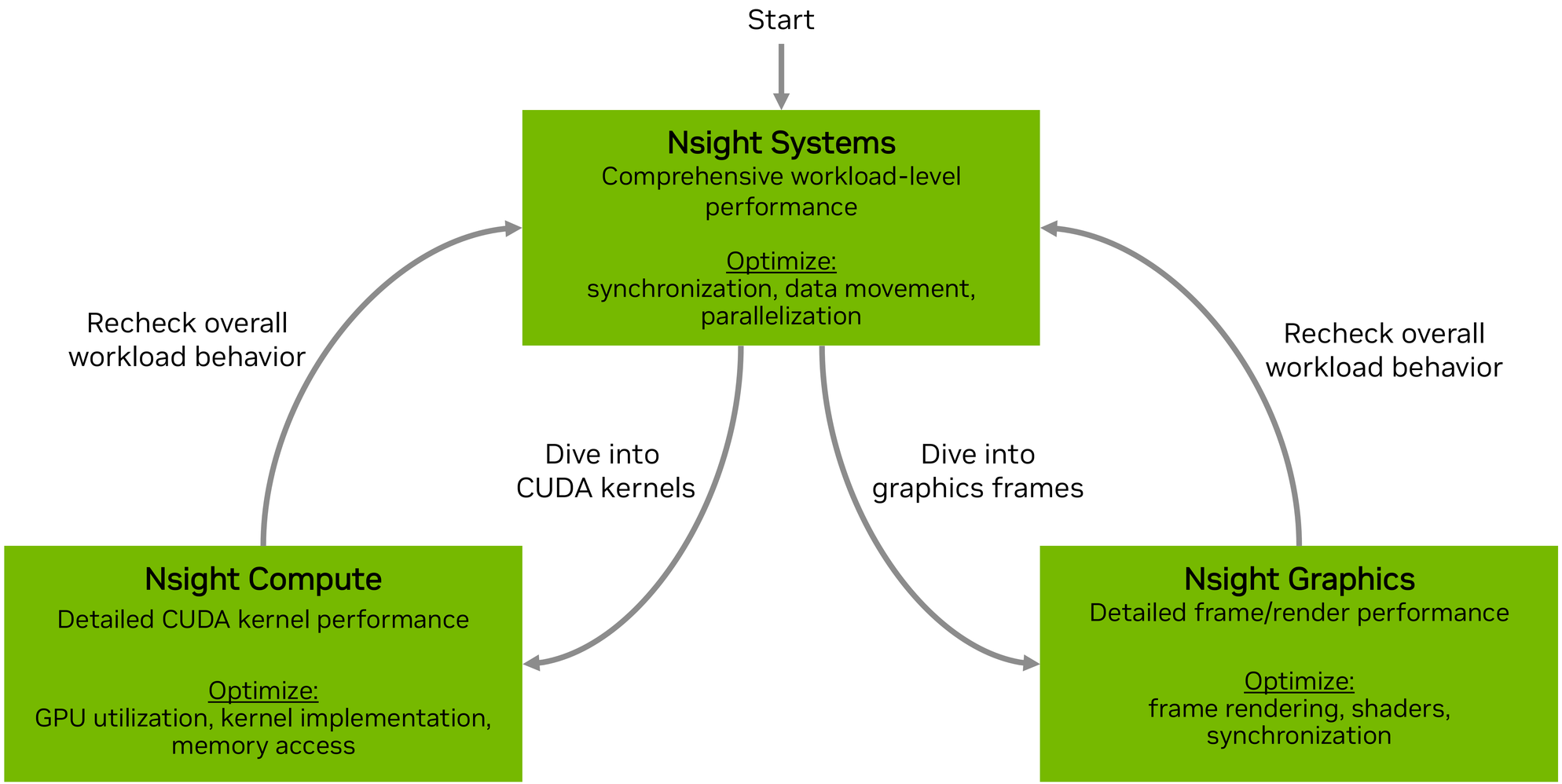

**Correctness Checker**: Compute Sanitizer

```
$ compute-sanitizer --leak-check full memcheck_demo
========= COMPUTE-SANITIZER
Mallocing memory
Running unaligned_kernel
Ran unaligned_kernel: no error
Sync: no error
Running out_of_bounds_kernel
Ran out_of_bounds_kernel: no error
Sync: no error
========= Invalid __global__ write of size 4 bytes
=========     at 0x60 in memcheck_demo.cu:6:unaligned_kernel(void)
=========     by thread (0,0,0) in block (0,0,0)
=========     Address 0x400100001 is misaligned
```

**Profilers**: Nsight Systems, Nsight Compute, CUPTI, NVIDIA Tools eXtension (NVTX)
Nsight Graphics



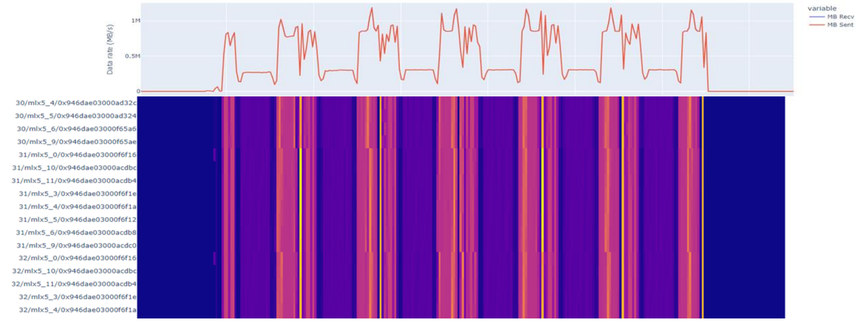
# NVIDIA Nsight Performance Analysis Tools

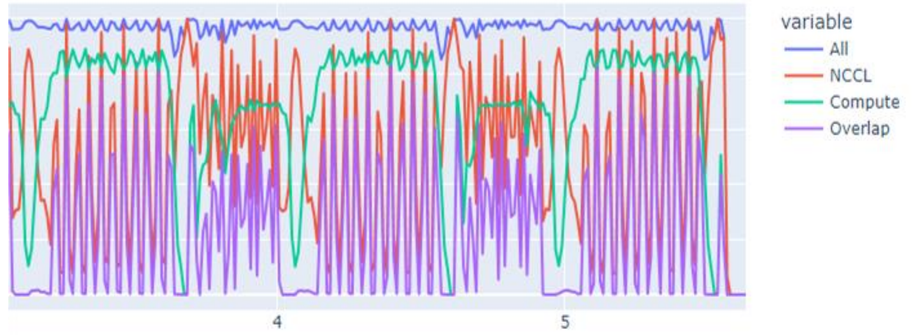


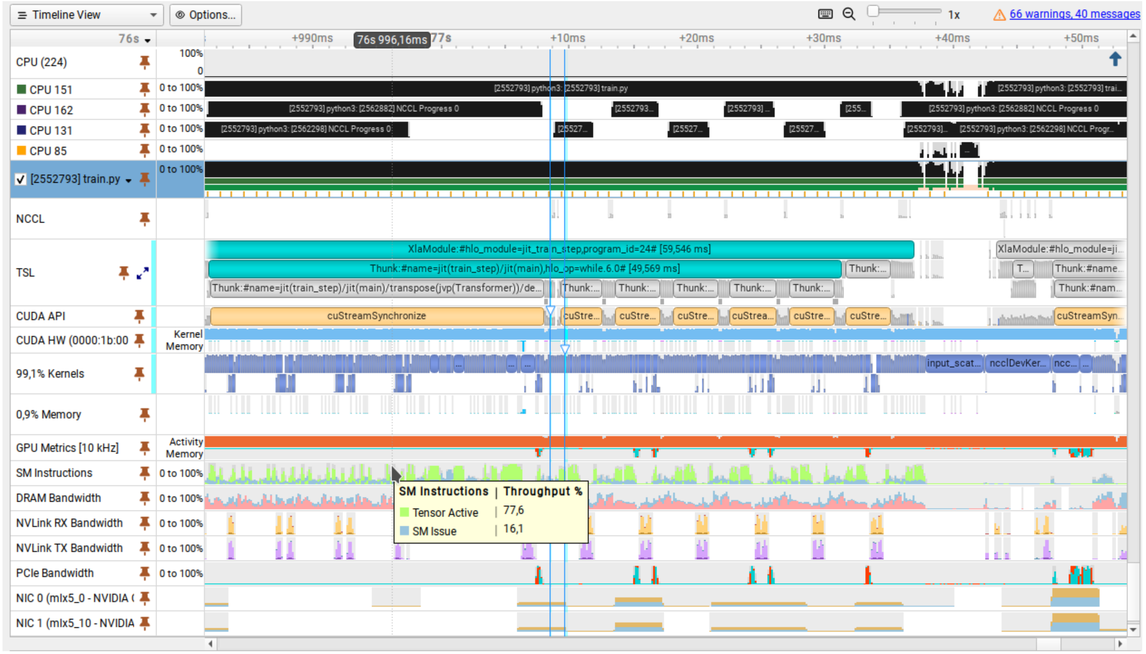
# Nsight Systems

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

- Application, system-wide and cluster-level profiling tool
- Analyze workload balancing across multiple GPUs, CPUs, DPUs, NICs, servers …
- Locate optimization opportunities
  - Idle processors (time & width)
  - Parallelism opportunities
  - Unnecessary synchronizations
- Timeline visualization
- Single & multi-report data analysis recipes in Python
- Data sources
  - API trace: CUDA, OpenACC, OpenMP, cuBLAS, …
  - Communication: MPI, NCCL, NVSHMEM, UCX, …
  - OS: thread state, core occupancy, OS runtime (pthread, glibc), syscalls, ftrace, …
  - PyTorch via NVTX
  - Sampling: GPU and CPU metrics, Python call stacks
  - Plug-ins → BYO data providers & data analysis recipes



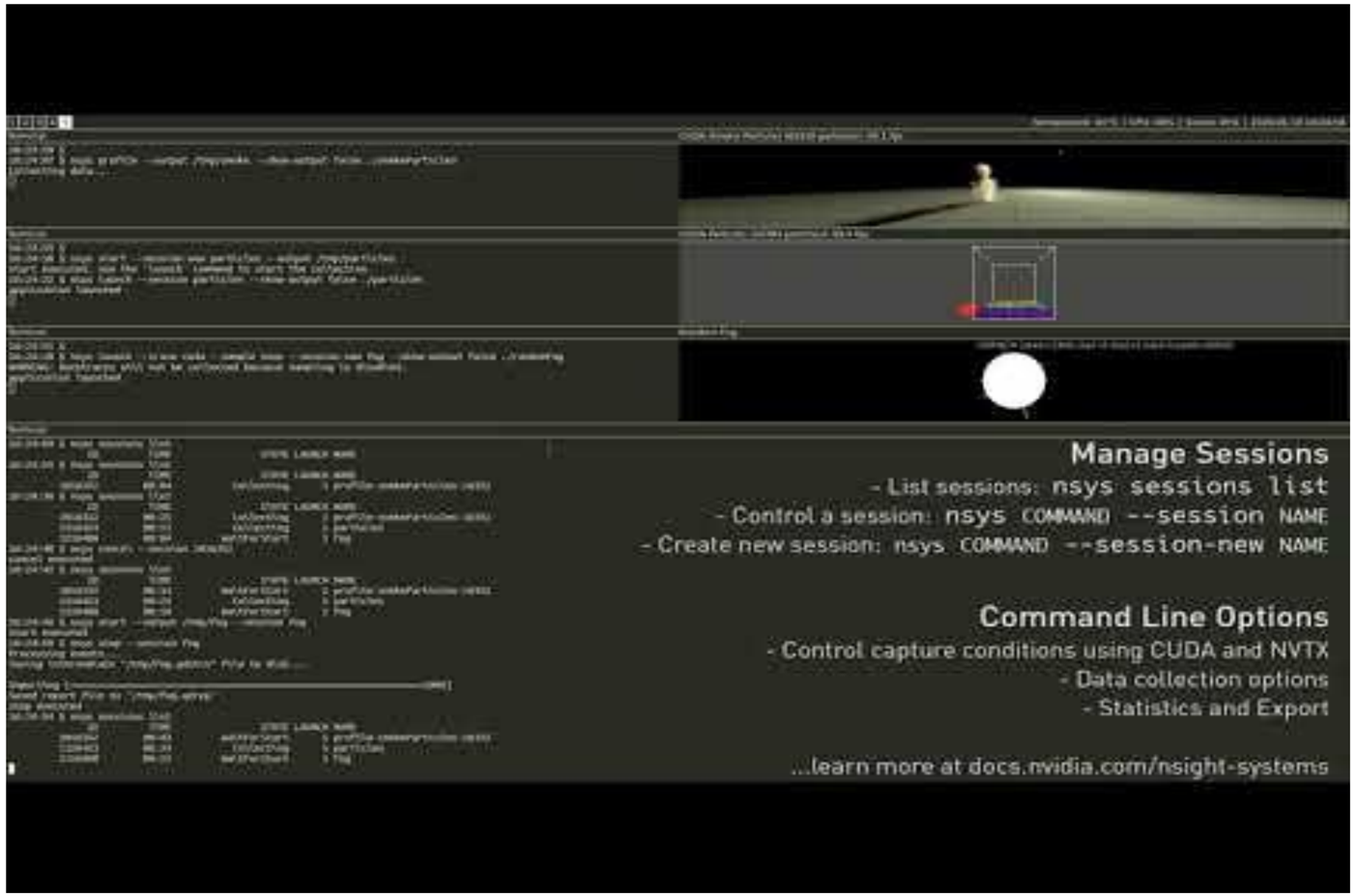



# Command Line Interface

The Nsight Systems CLI provides several commands. See `nsys --help`.

- Basic profiling session
  `nsys profile`
- Interactive sessions (scriptable)
  `nsys launch | start | stop | cancel`
  `nsys sessions list`
  `nsys status | shutdown`
- Statistics and export
  `nsys stats`, `nsys export`
  (export to sqlite, hdf, text, json, info, arrow, arrowdir, parquetdir)
- Analysis
  `nsys analyze`, `nsys recipe`




https://youtu.be/r2ewwd4d0vc

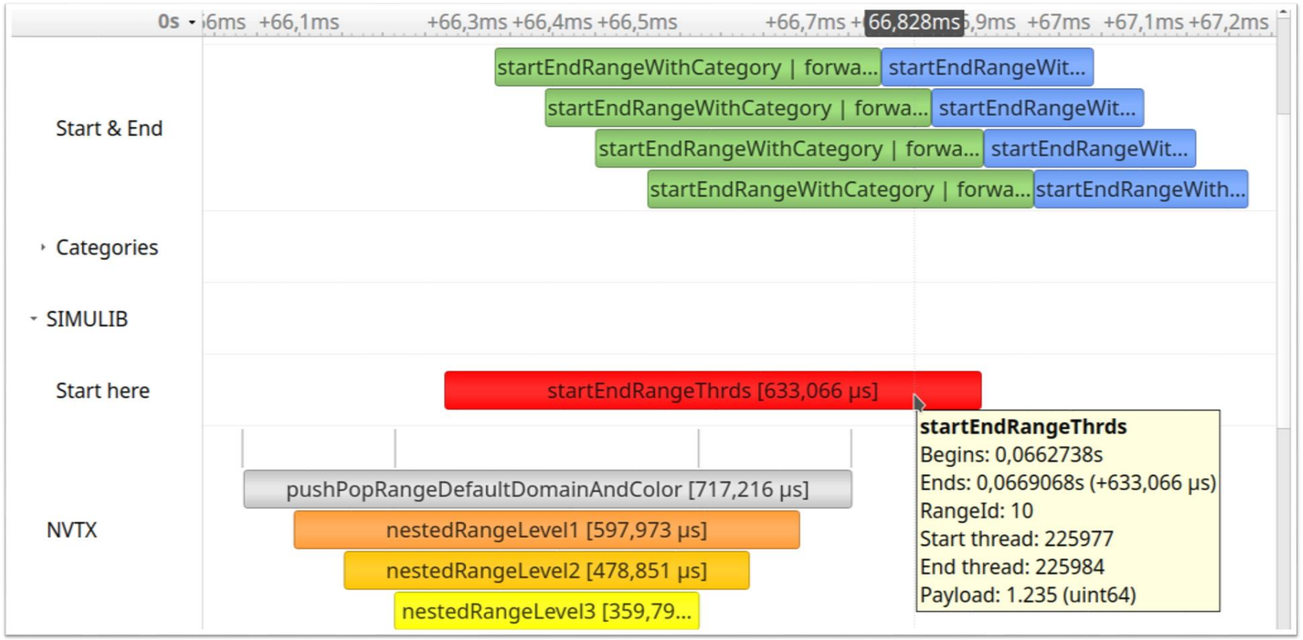
# NVTX

# NVIDIA Tools Extensions (NVTX)

C, C++, Python

## Annotate application source code

- Header-only library
  ```
  #include <nvtx3/nvToolsExt.h>
  ```
- **Marker**
  ```
  nvtxMark("Point in time");
  ```
- **Push/Pop Range**
  ```
  nvtxRangePush("Perfectly nested range");
  // Do something interesting in the range.
  nvtxRangePop();
  ```

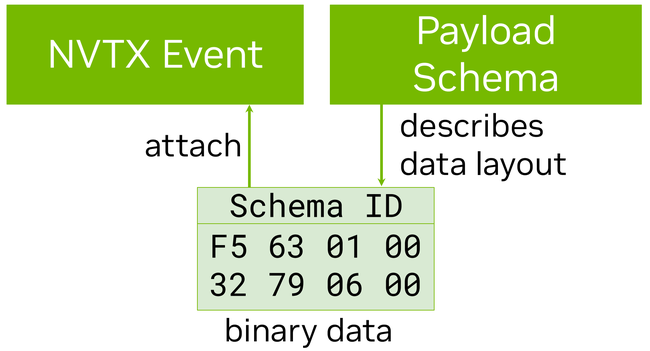


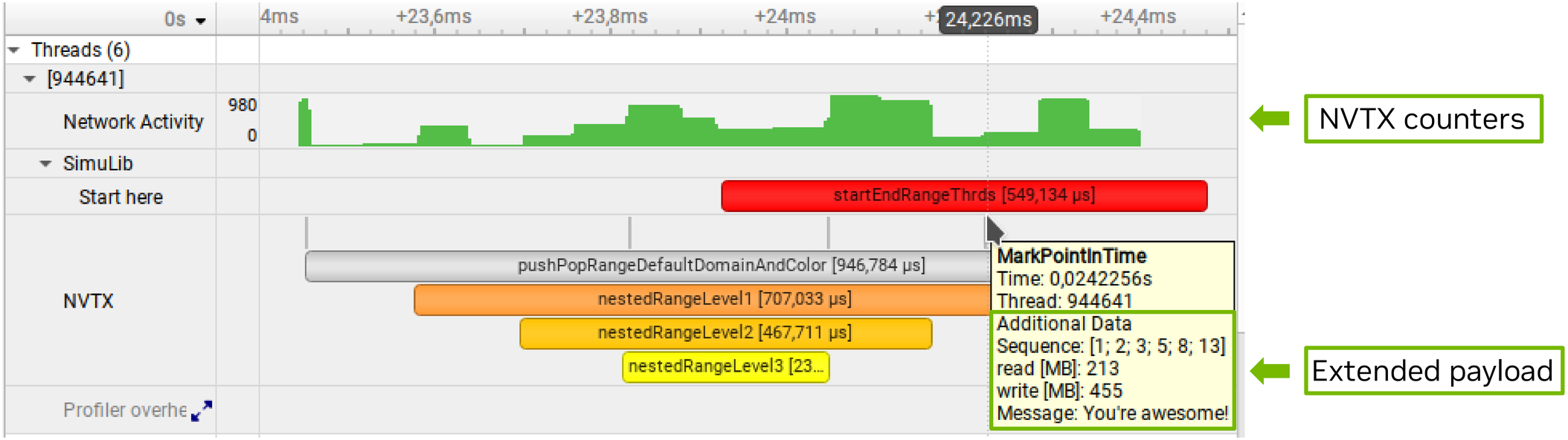

- **Start/End Range**
  ```
  nvtxRangeId_t rangeId = nvtxRangeStart("This is a start/end range");
  // Somewhere else in the code, not necessarily same thread as range start call
  nvtxRangeEnd(rangeId);
  ```
- Provides concepts like domains, categories, colors, registered strings, and extended payloads
- Sources: https://github.com/NVIDIA/NVTX/tree/release-v3/
- Enable Nsight Systems NVTX collection with `--trace nvtx` (enabled by default)

# NVTX Extended Payloads, Counters & Nsight Systems Plugins

Build your own data collectors!

- Pass arbitrary data to NVTX events. Define the layout of this data without additional data conversion.
- NVTX counters
  - Data layout of counter groups via extended payload schemas
  - User-defined scopes
- Nsight Systems plugins
  - Additional processes that can be used to collect data by emitting NVTX events
  - `nsys profile|start --enable PluginName,flag1,arg1=val1,...`




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

# Case Study 1: Video Background Blurring Pipeline

# Video Segmentation Pipeline

## Overview

### Video Segmentation for Background Blurring

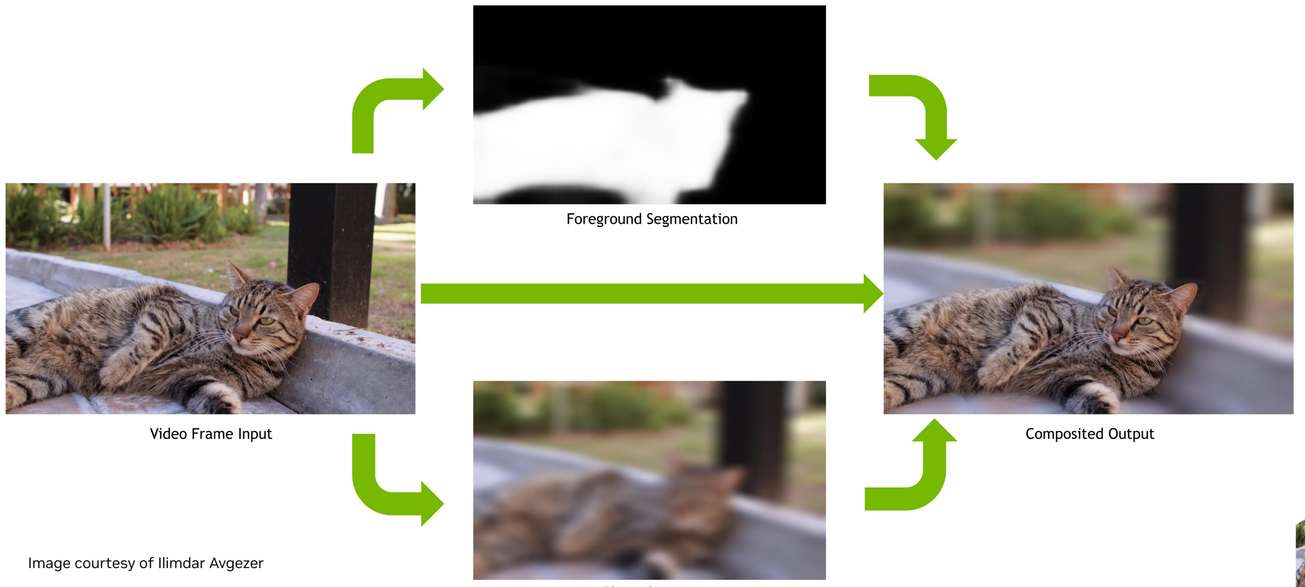



Image courtesy of Ilimdar Avgezer

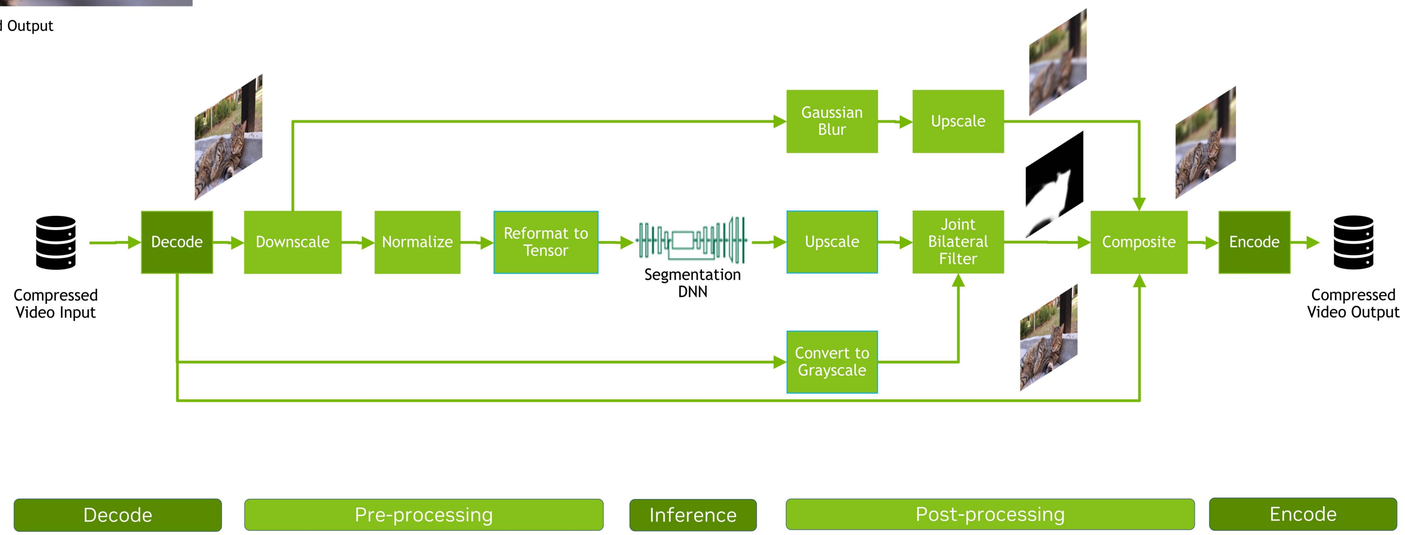
Python baseline code uses the following components:

- ffmpeg for the decoding and encoding steps
- OpenCV for the pre-processing and post-processing
- PyTorch for the segmentation DNN.

The 474 frames are grouped into batches of four.

### Video Segmentation Pipeline

Overview of the End-to-End Sample Pipeline




Every frame goes through five steps

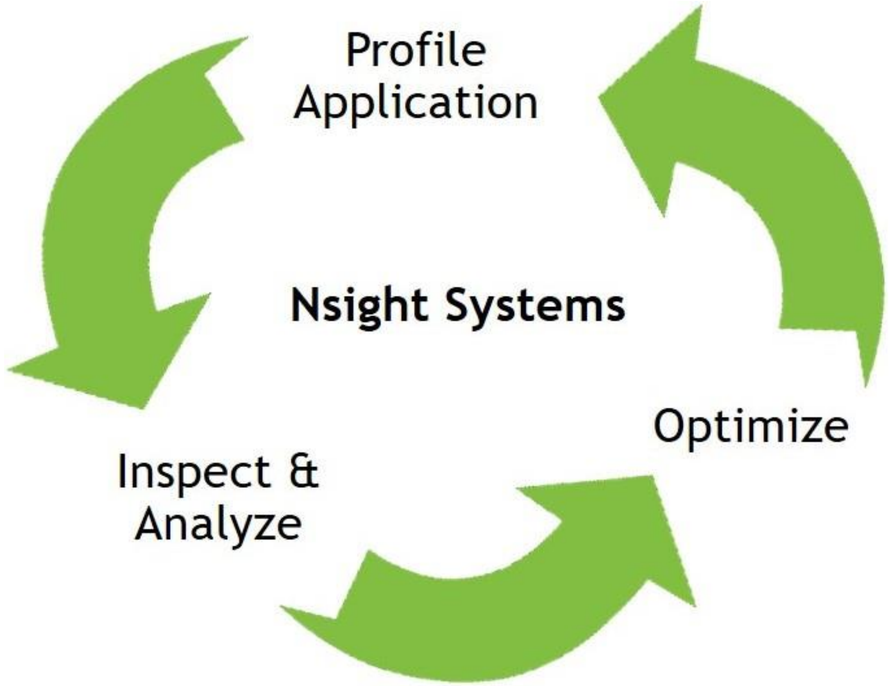
# Video Segmentation Pipeline

## Optimization Workflow

1. Add NVTX annotations to the code
2. Profile the application to get a baseline profile

```
nsys profile \
  --trace cuda,nvtx,osrt \
  --output reports/baseline \
  --force-overwrite=true \
python video_segmentation/main_nvtx.py
```

3. Analyze the profile
4. Optimize the code to address the bottleneck



# Multi-Report Analysis

## Video Segmentation Pipeline

Goal: Find the fastest pipeline and the most time-consuming step for different batch sizes

- Create profiles for different batch sizes

```
for bs in 1 2 4 8 16 32; do \
  printf -v bs_zfill "%02d" ${bs}; \
  nsys profile --trace cuda,nvtx -f true \
    -o reports/cvcuda_batchsize/cvcuda_bs${bs_zfill} \
  python video_segmentation/main_nvtx-cvcuda-nvcodec.py
${bs} ; \
done
```

- Install recipe system:
  https://docs.nvidia.com/nsight-systems/InstallationGuide/index.html#installing-advanced-analysis-system
- Which recipes are available?

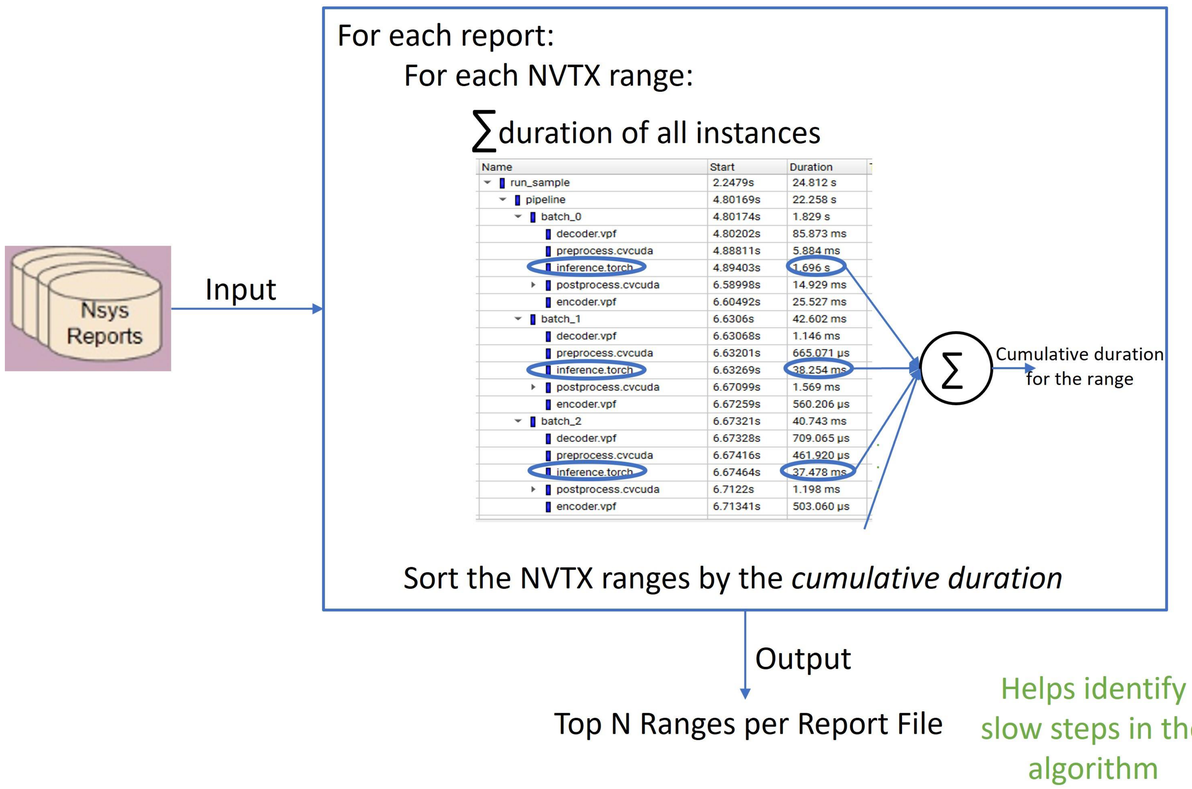
```
nsys recipe --help
```

# Multi-Report Analysis

## NVTX GPU Projection Summary Recipe

- Goal: find the fastest pipeline and the most time-consuming step (most valuable optimization target)

```
nsys recipe nvtx_gpu_proj_sum \
  --input reports/cvcuda_batchsize \
  --output reports/cvcuda_batchsize/results_nvtx_gpu_proj_sum \
  --force-overwrite \
  --log-level=error
```

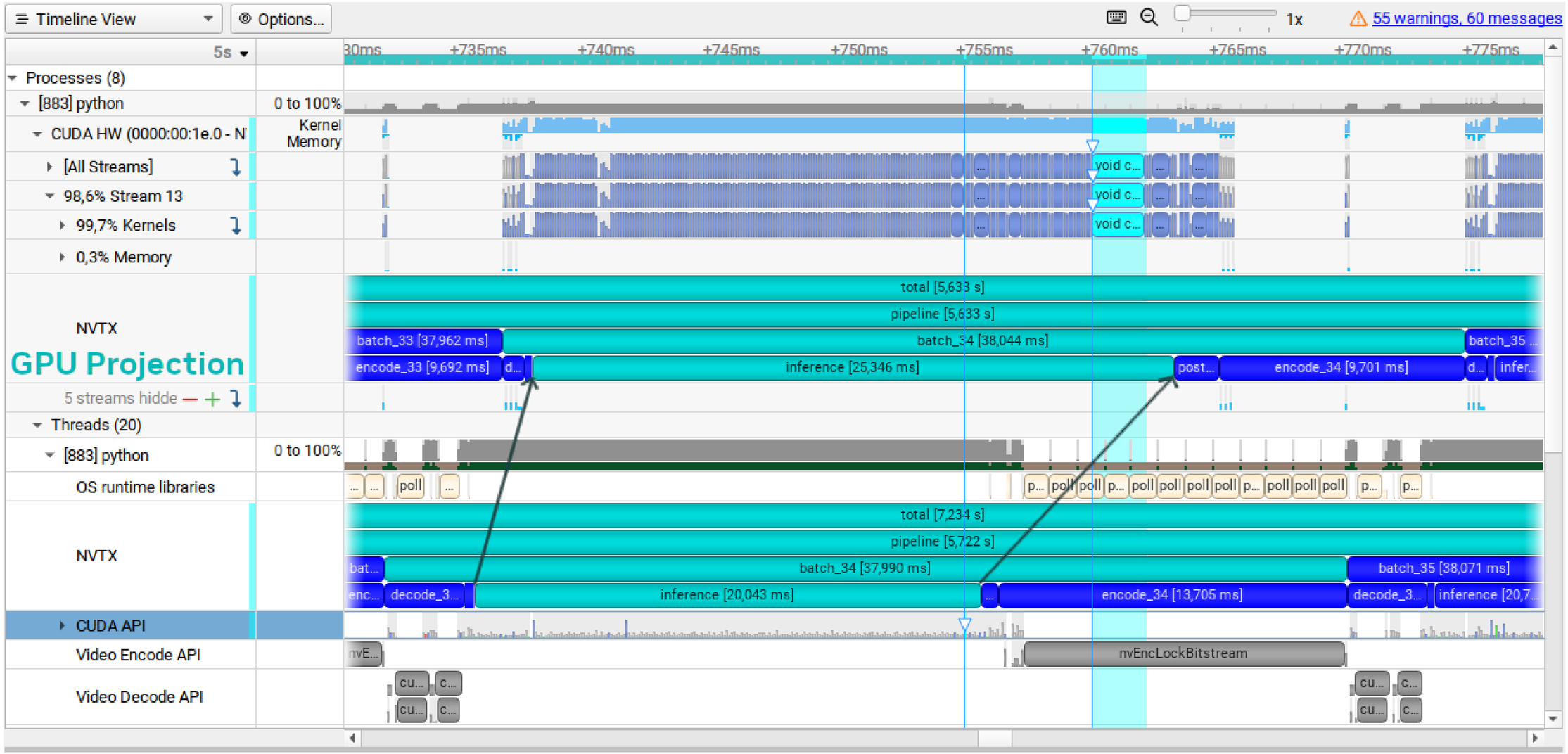




```
nsys recipe nvtx_gpu_proj_sum --help
```

# NVTX GPU Projection

## Video Segmentation Pipeline



# NVTX GPU Projection Summary Recipe

CPU

| File | cvcuda_bs01 | cvcuda_bs02 | cvcuda_bs04 | cvcuda_bs08 | cvcuda_bs16 |
|---|---|---|---|---|---|
| NVTX Range | | | | | |
| total | 12.1 | 7.9 | 7.2 | 7.8 | 8.1 |
| pipeline | 10.6 | 6.4 | 5.7 | 6.3 | 6.5 |
| inference | 9.5 | 5.5 | 3.2 | 2.0 | 1.5 |
| encode | 0.2 | 0.1 | 1.7 | 3.4 | 4.1 |
| batch_0 | 1.1 | 1.1 | 1.1 | 1.2 | 1.3 |

GPU projected

| File | cvcuda_bs01 | cvcuda_bs02 | cvcuda_bs04 | cvcuda_bs08 | cvcuda_bs16 |
|---|---|---|---|---|---|
| NVTX Range | | | | | |
| total | 11.4 | 7.1 | 6.4 | 7.1 | 7.3 |
| pipeline | 10.6 | 6.3 | 5.6 | 6.3 | 6.5 |
| inference | 9.5 | 5.5 | 3.9 | 3.8 | 3.6 |
| encode | 0.1 | 0.1 | 1.1 | 1.7 | 2.0 |
| batch_0 | 1.0 | 1.1 | 1.1 | 1.1 | 1.2 |

# Case Study 2: MLPerf DeepCAM AI Benchmark

# MLPerf DeepCAM Benchmark

## Data Collection

- Nsight Systems is not aware of any cluster schedulers such as Slurm

```
srun <srun args> nsys profile <nsys args> -o report_name.%q{SLURM_PROCID} your_application
```

- Limit the amount of recorded data
  - `--capture-range`
    Limit the interval in which data is collected. (Capture ranges can be triggered via the CUDA profiler API and NVTX events.)
  - `--duration`
    Set a collection duration.
  - `--delay`
    Delay data recording, e.g. to skip the initialization phase.
  - `--nvtx-domain-[include|exclude]`
    NVTX domain filtering can include or exclude events from an NVTX domain
  - Record only a set of ranks or just one, e.g., with a helper script

```
#/bin/bash
if [ $SLURM_LOCALID -eq 0 ]; then
   nsys profile "$@"
else
   "$@"
fi
```

# MLPerf DeepCAM Benchmark

## Data Collection and NCCL Utilization Recipe

- The reports have been collected with the following commands/script:

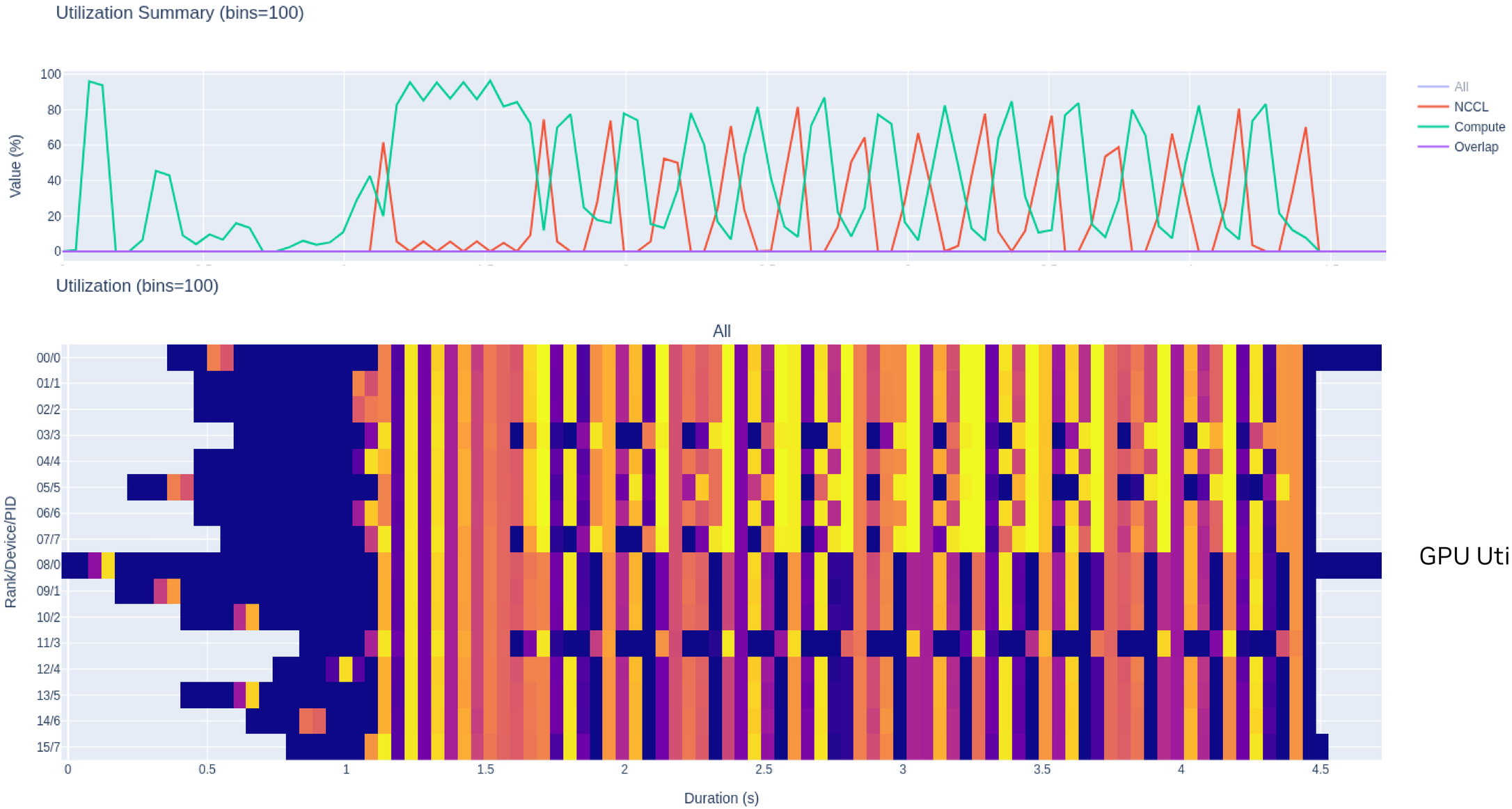
```
#/bin/bash
# Apply system-wide options only to one rank per node
if [ $SLURM_LOCALID -eq 0 ]; then
    nsys_system_wide="--nic-metrics=true --storage-metrics=--storage-devices=all"
fi

srun -n 16 <more srun arguments> \
    nsys profile ${nsys_system_wide} --trace=cuda,nvtx,osrt \
      --cuda-graph-trace=node \
      --pytorch=autograd-nvtx \
      --capture-range=cudaProfilerApi --kill=none \
      -o ${OUTPUT_DIR}/mlperf_...${SLURM_PROCID} -f true \
    /usr/bin/python ./train.py --capture_range_start 11 --capture_range_stop 29 <more training parameters>
```

- Run NCCL GPU Time Utilization Recipe

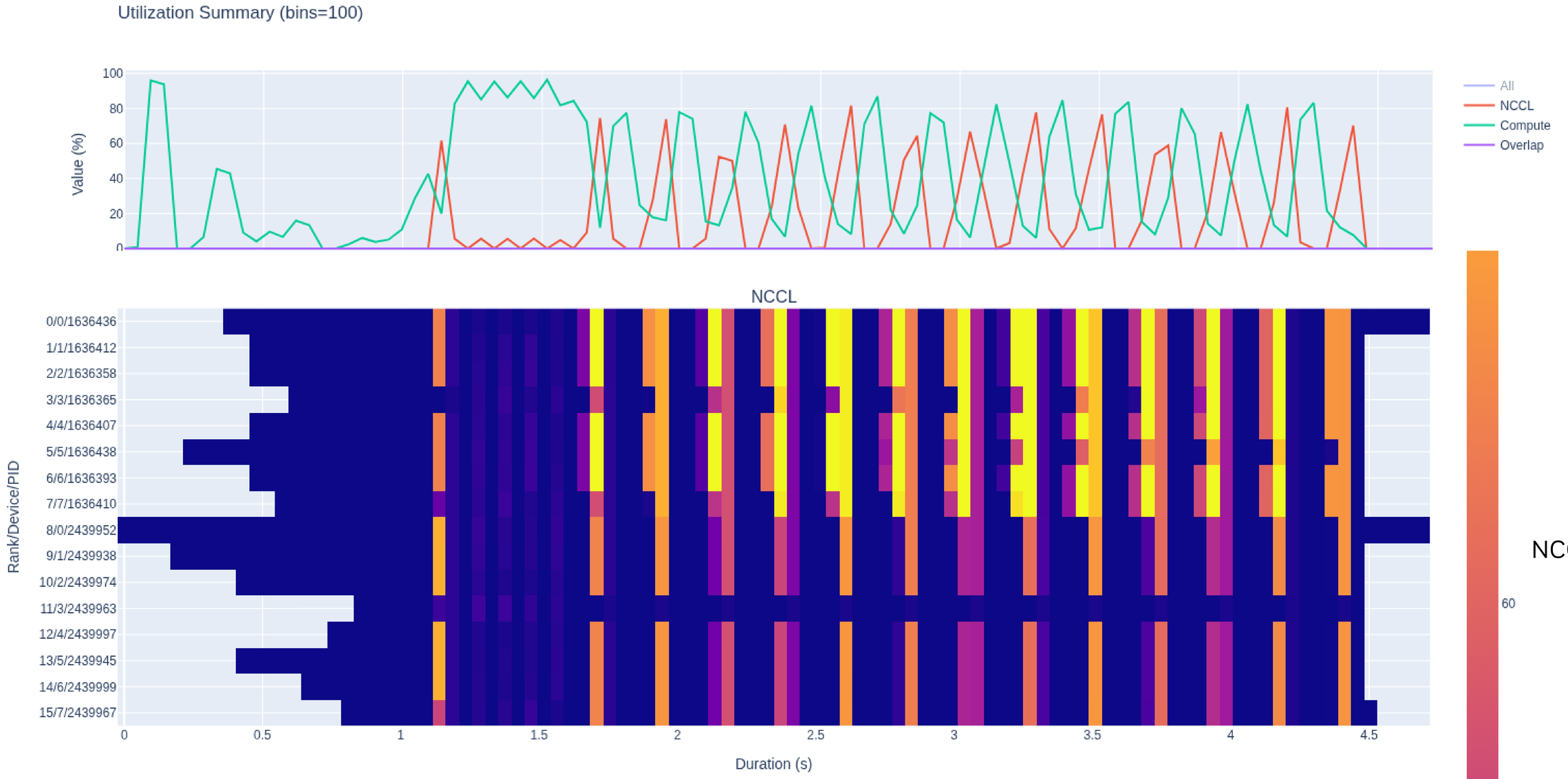
```
nsys recipe nccl_gpu_time_util_map
    --input /dli/task/reports/precollected/mlperf/initial_run \
    --output /dli/task/reports/precollected/mlperf/initial_run/results_nccl_gpu_util_map \
    --force-overwrite \
    --bins 100 \
    --log-level=error \
```

# MLPerf DeepCAM– Initial Run




GPU Utilization

# MLPerf DeepCAM– Initial Run

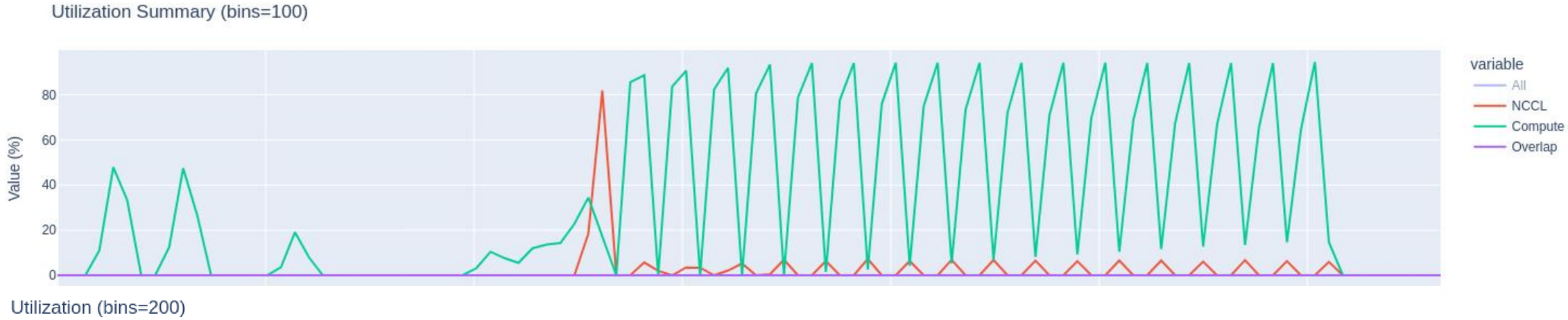


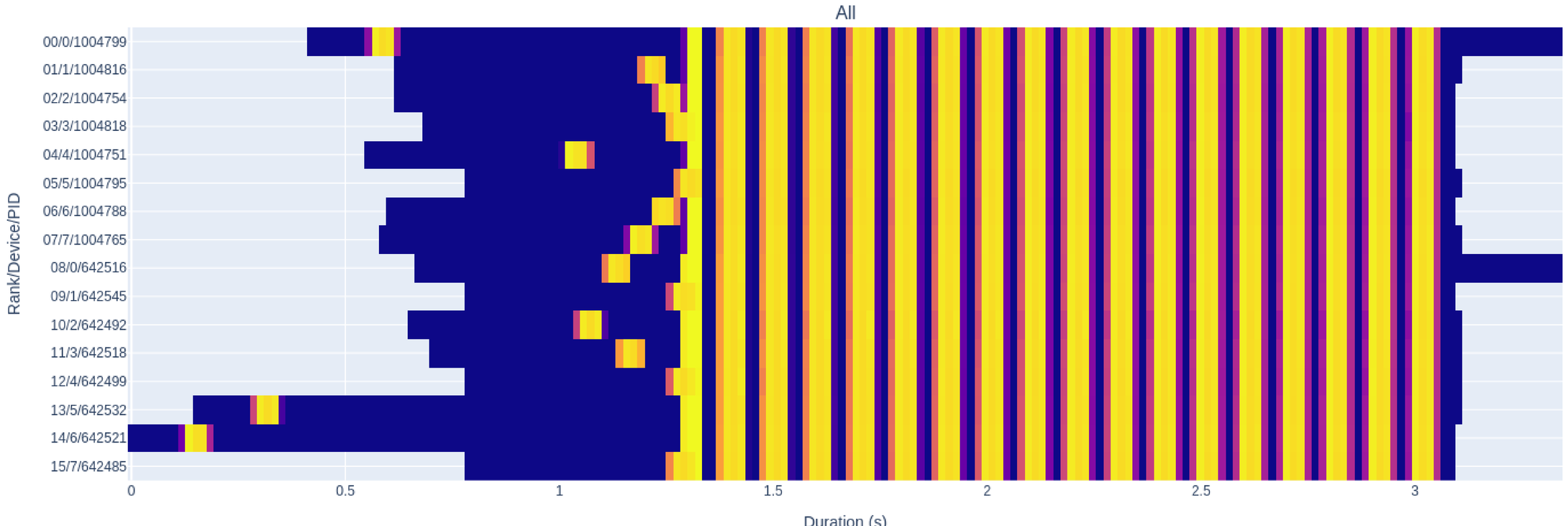

NCCL Kernels



NVIDIA

# MLPerf DeepCAM – Improved I/O



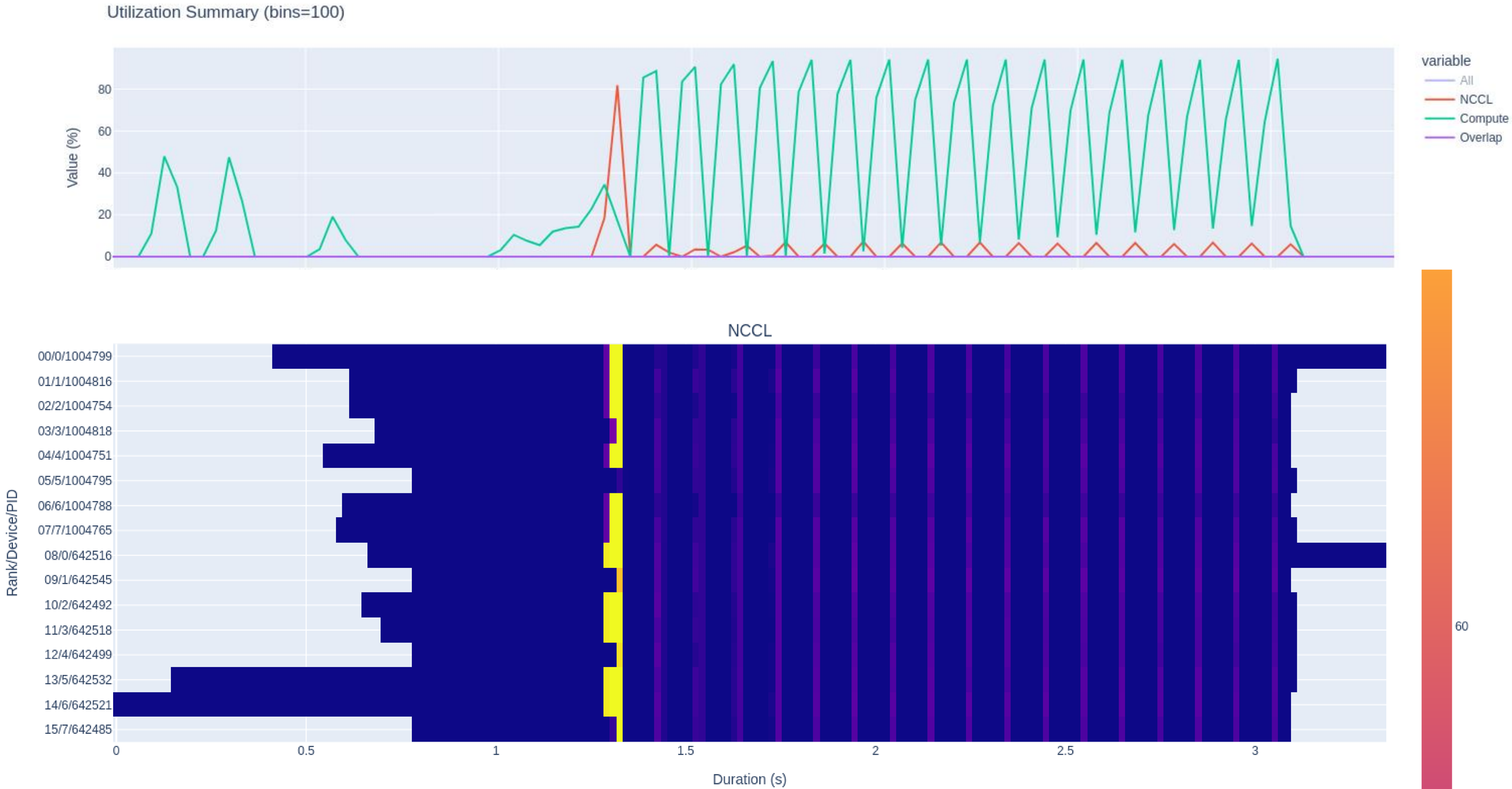




GPU Utilization



NVIDIA

# MLPerf DeepCAM – Improved I/O




NCCL Kernels

# THANK YOU!

| | |
|---|---|
| Download | https://developer.nvidia.com/nsight-systems<br>**NOTE**: website version is newer than CUDA Toolkit version |
| Docs | https://docs.nvidia.com/nsight-systems/index.html |
| Forums | https://devtalk.nvidia.com/default/board/308/nsight-systems/ |
| Email | nsight-systems@nvidia.com |
| Blogs | https://developer.nvidia.com/blog/nvidia-nsight-systems-containers-cloud<br>https://developer.nvidia.com/blog/nsight-systems-exposes-gpu-optimization<br>https://developer.nvidia.com/blog/understanding-the-visualization-of-overhead-and-latency-in-nsight-systems<br>https://developer.nvidia.com/blog/nvidia-tools-extension-api-nvtx-annotation-tool-for-profiling-code-in-python-and-c-c/ |
| Trainings | https://www.youtube.com/playlist?list=PL5B692fm6--ukF8S7ul5NmceZhXLRv_lR |