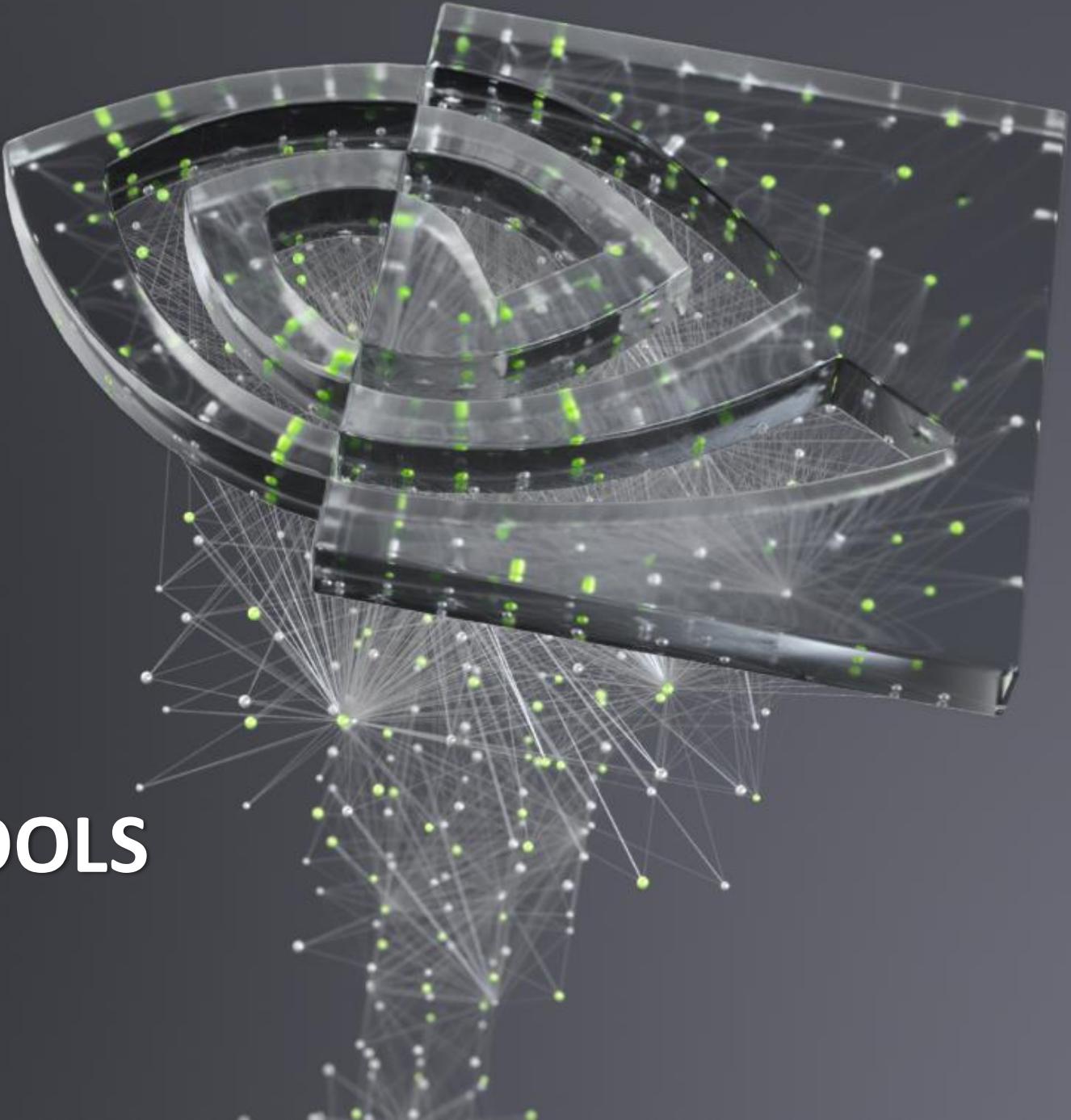




# NVIDIA DEVELOPER TOOLS

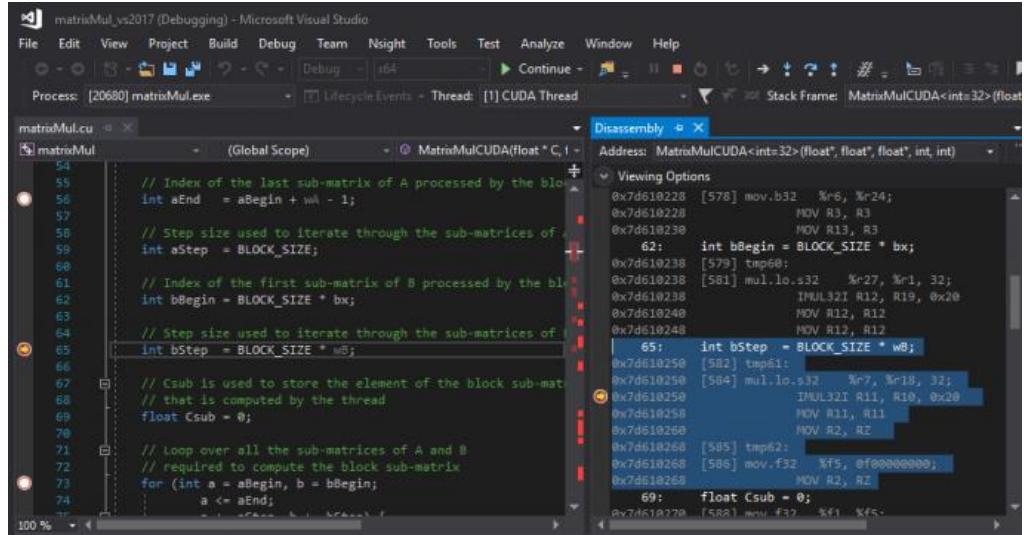
Robert Searles

Senior Solutions Architect, NVIDIA

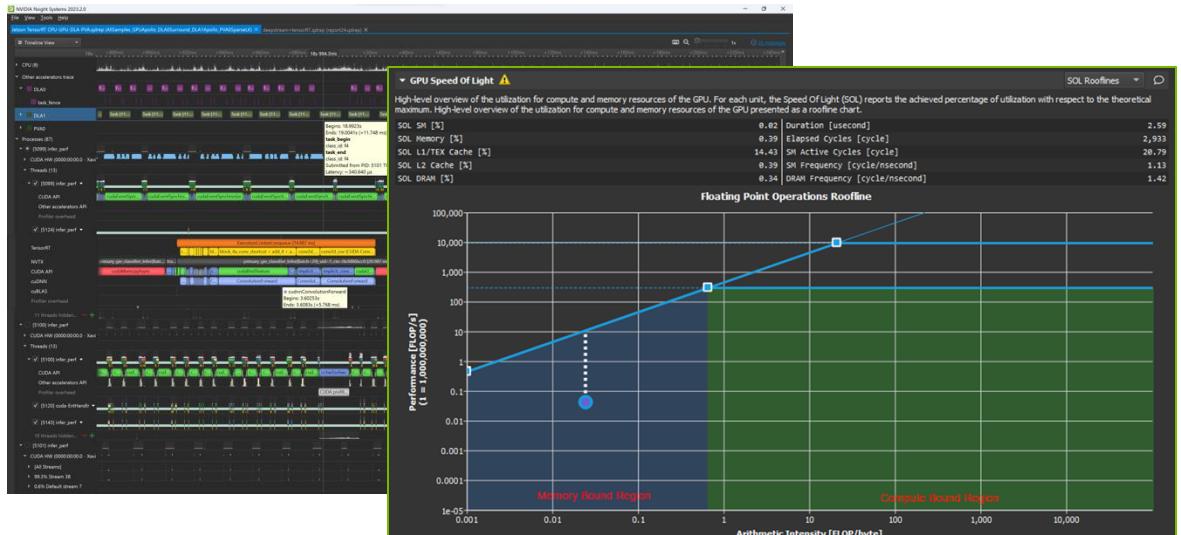


# Developer Tools Ecosystem

**Debuggers:** cuda-gdb, Nsight Visual Studio Edition Nsight Visual Studio Code Edition



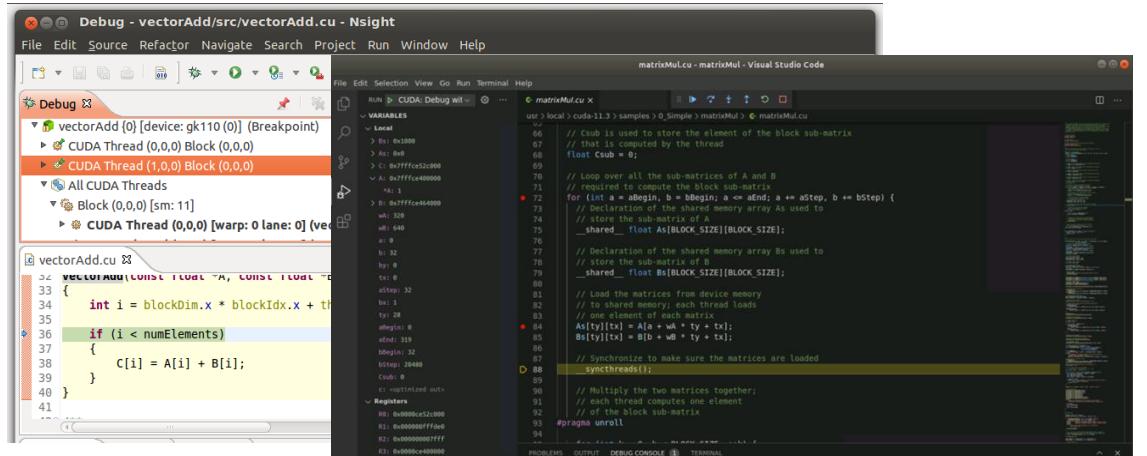
**Profilers:** Nsight Systems, Nsight Compute, CUPTI, [NVIDIA Tools eXtension \(NVTX\)](#)



**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
```

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



# Programming the NVIDIA Platform

CPU, GPU, and Network

## ACCELERATED STANDARD LANGUAGES

ISO C++, ISO Fortran

```
std::transform(par, x, x+n, y, y,
    [=](float x, float y) { return y +
a*x; })
;
```

```
do concurrent (i = 1:n)
    y(i) = y(i) + a*x(i)
enddo
```

```
import cunumeric as np
...
def saxpy(a, x, y):
    y[:] += a*x
```

## INCREMENTAL PORTABLE OPTIMIZATION

OpenACC, OpenMP

```
#pragma acc data copy(x,y) {
...
std::transform(par, x, x+n, y, y,
    [=](float x, float y) {
        return y + a*x;
});
...
}

#pragma omp target data map(x,y) {
...
std::transform(par, x, x+n, y, y,
    [=](float x, float y) {
        return y + a*x;
});
...
}
```

## PLATFORM SPECIALIZATION

CUDA

```
__global__
void saxpy(int n, float a,
            float *x, float *y) {
    int i = blockIdx.x*blockDim.x +
            threadIdx.x;
    if (i < n) y[i] += a*x[i];
}

int main(void) {
    ...
    cudaMemcpy(d_x, x, ...);
    cudaMemcpy(d_y, y, ...);

    saxpy<<< (N+255)/256, 256>>>(...);

    cudaMemcpy(y, d_y, ...);
}
```

## ACCELERATION LIBRARIES

Core

Math

Communication

Data Analytics

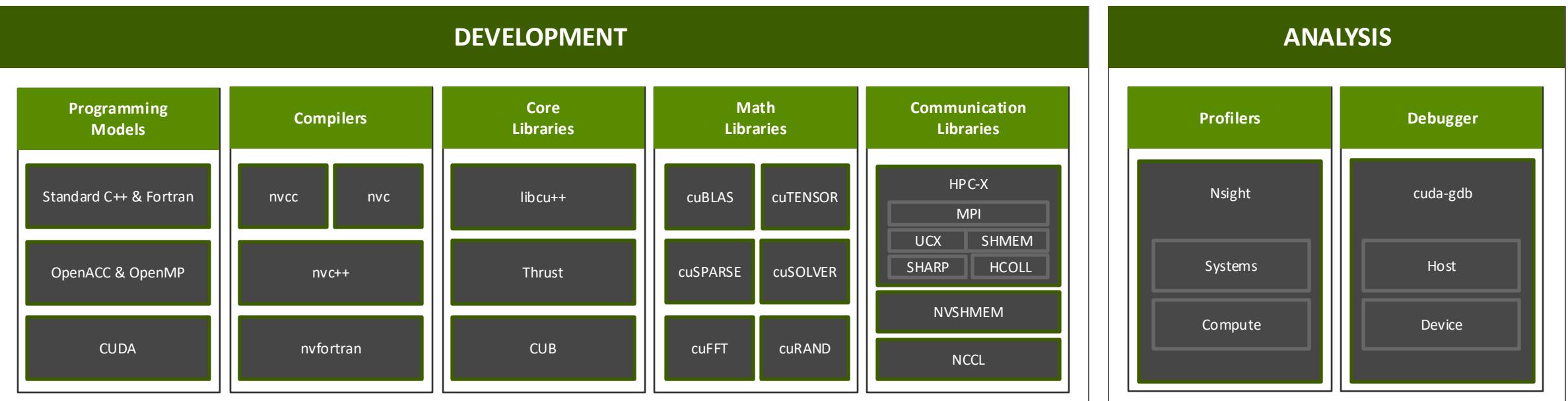
AI

Quantum



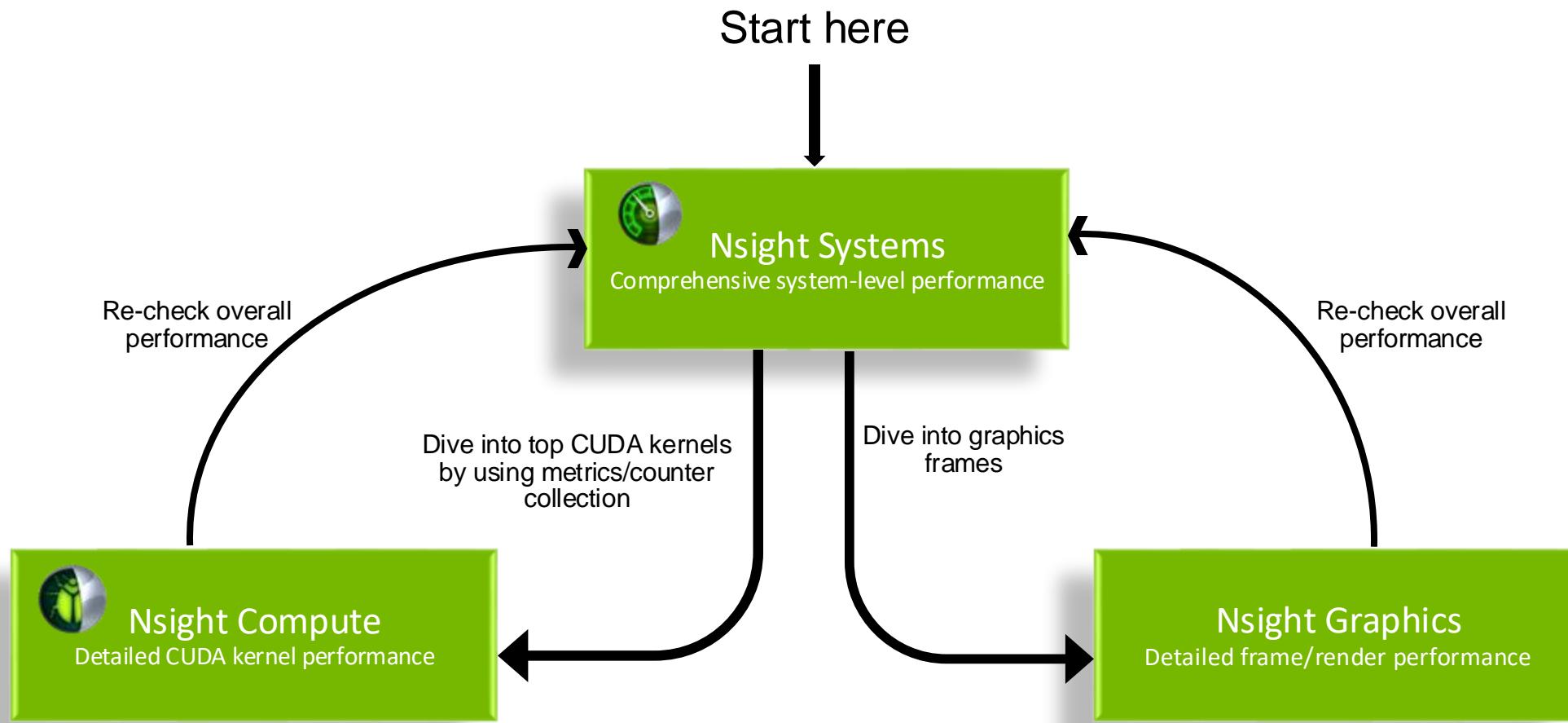
# NVIDIA HPC SDK

Available at [developer.nvidia.com/hpc-sdk](https://developer.nvidia.com/hpc-sdk), on NGC, via Spack, and in the Cloud



Develop for the NVIDIA Platform: GPU, CPU and Interconnect  
Libraries | Accelerated C++ and Fortran | Directives | CUDA  
7-8 Releases Per Year | Freely Available

# NSIGHT Profilers Workflow



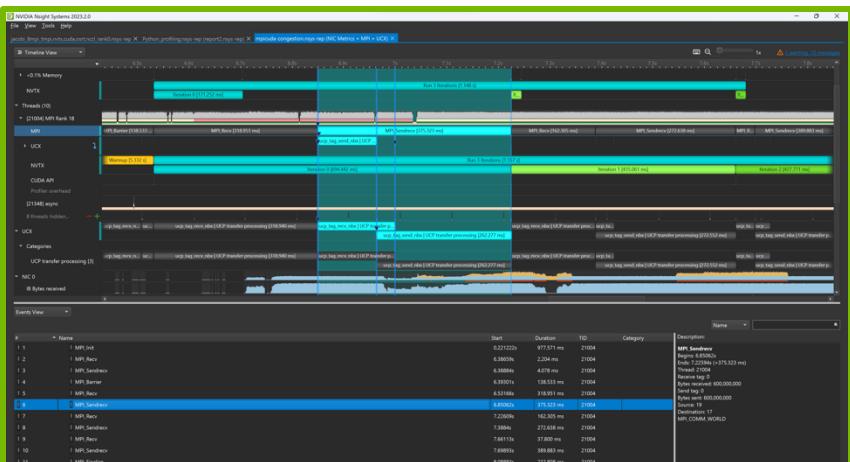
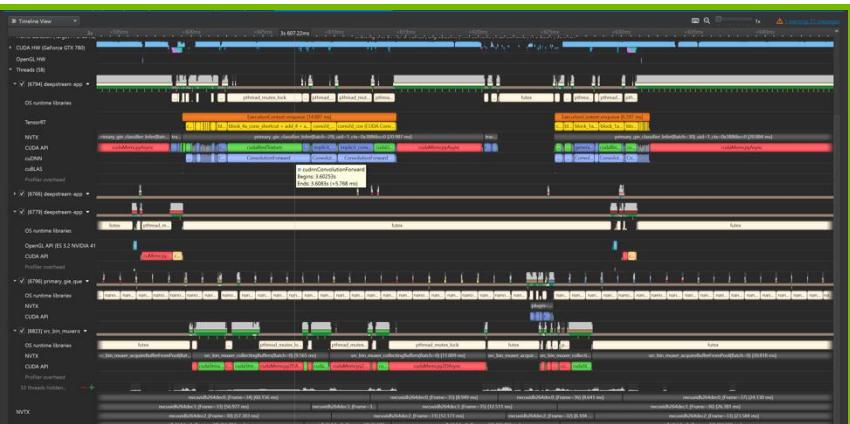


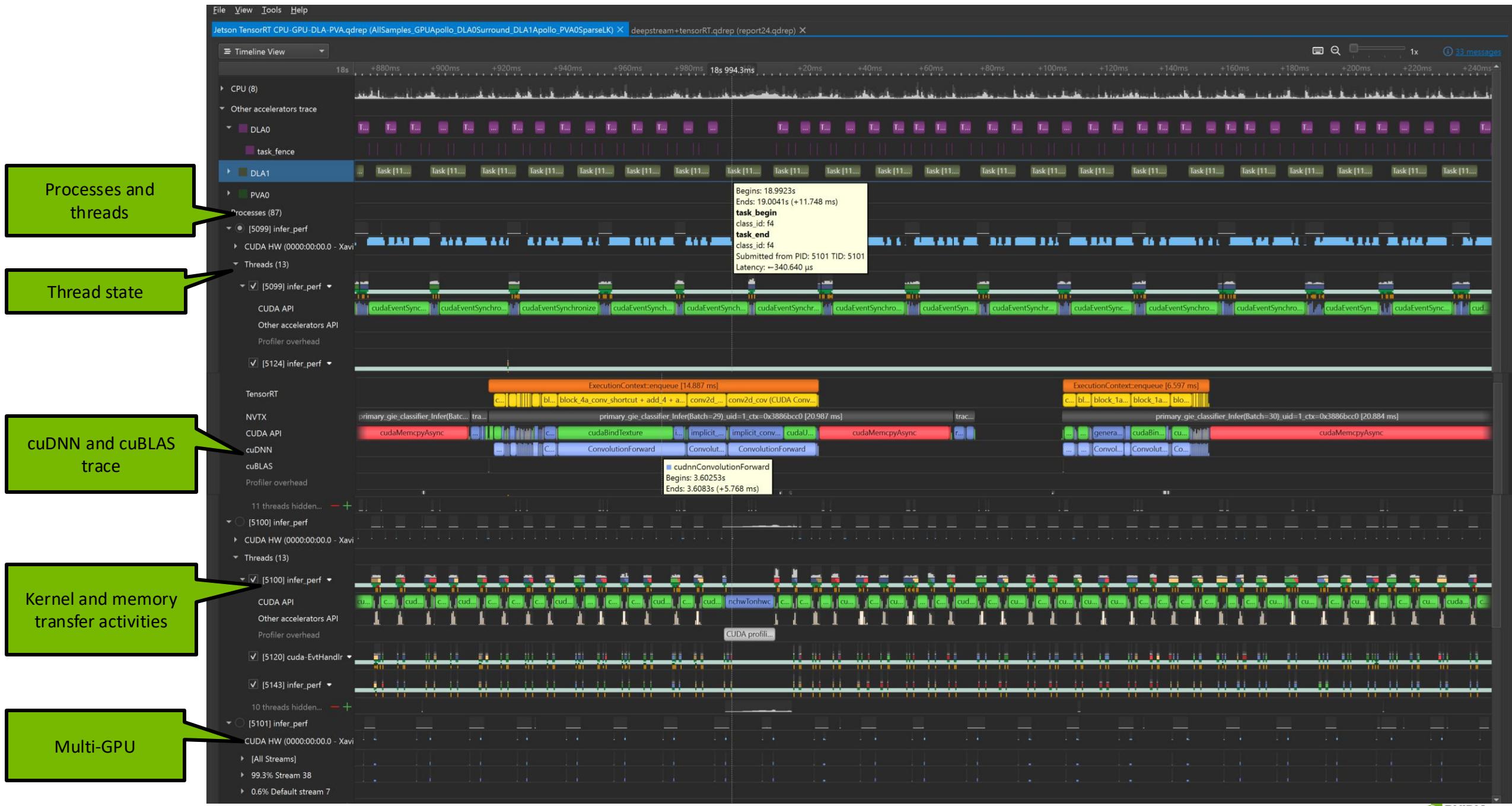
# Nsight Systems

## System Profiler

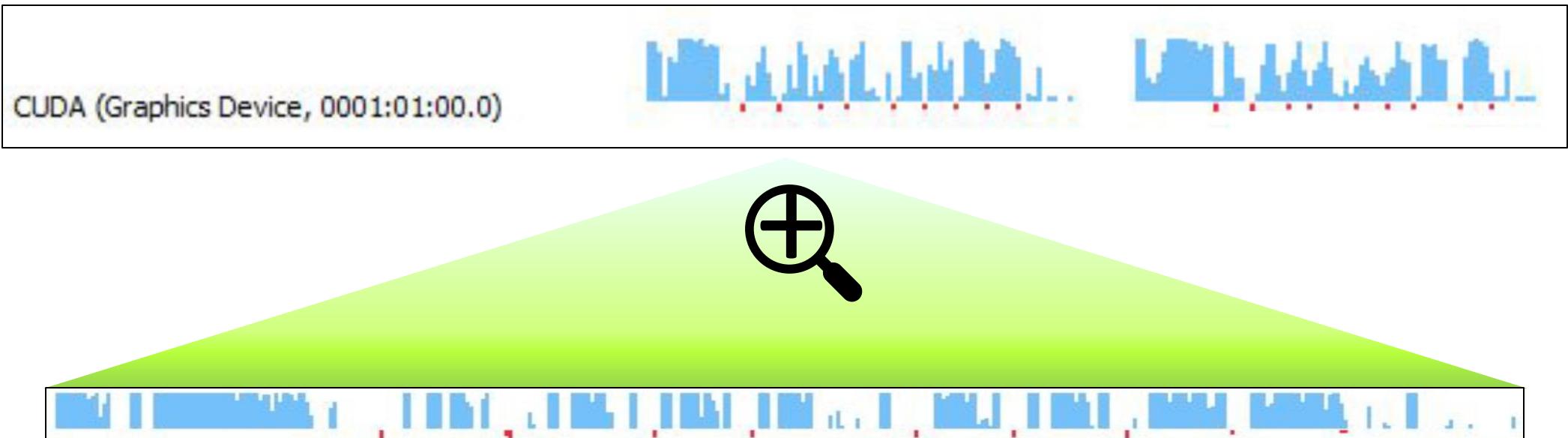
### Key Features:

- System-wide application algorithm tuning
  - Multi-process tree support
- Locate optimization opportunities
  - Visualize millions of events on a very fast GUI timeline
  - Identify 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
- Command Line, Standalone, IDE Integration
- OS: Linux (x86, ARM Server, Tegra), Windows, macOS X (host)
- GPUs: Pascal+
- Docs/product: <https://developer.nvidia.com/nsight-systems>



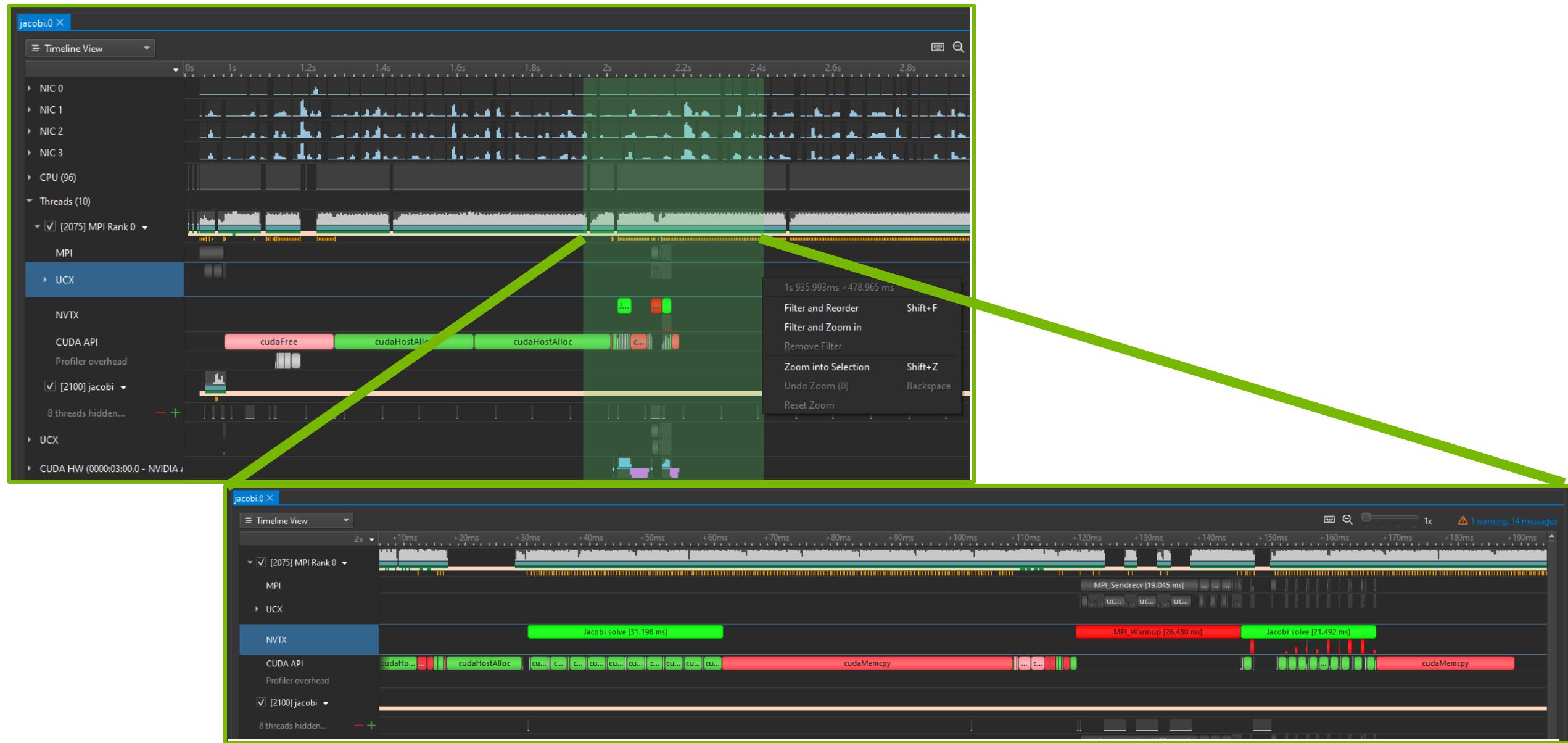


## Reading Utilization



- Zoom in to valleys to find gaps!

# Zoom/Filter to Exact Areas of Interest

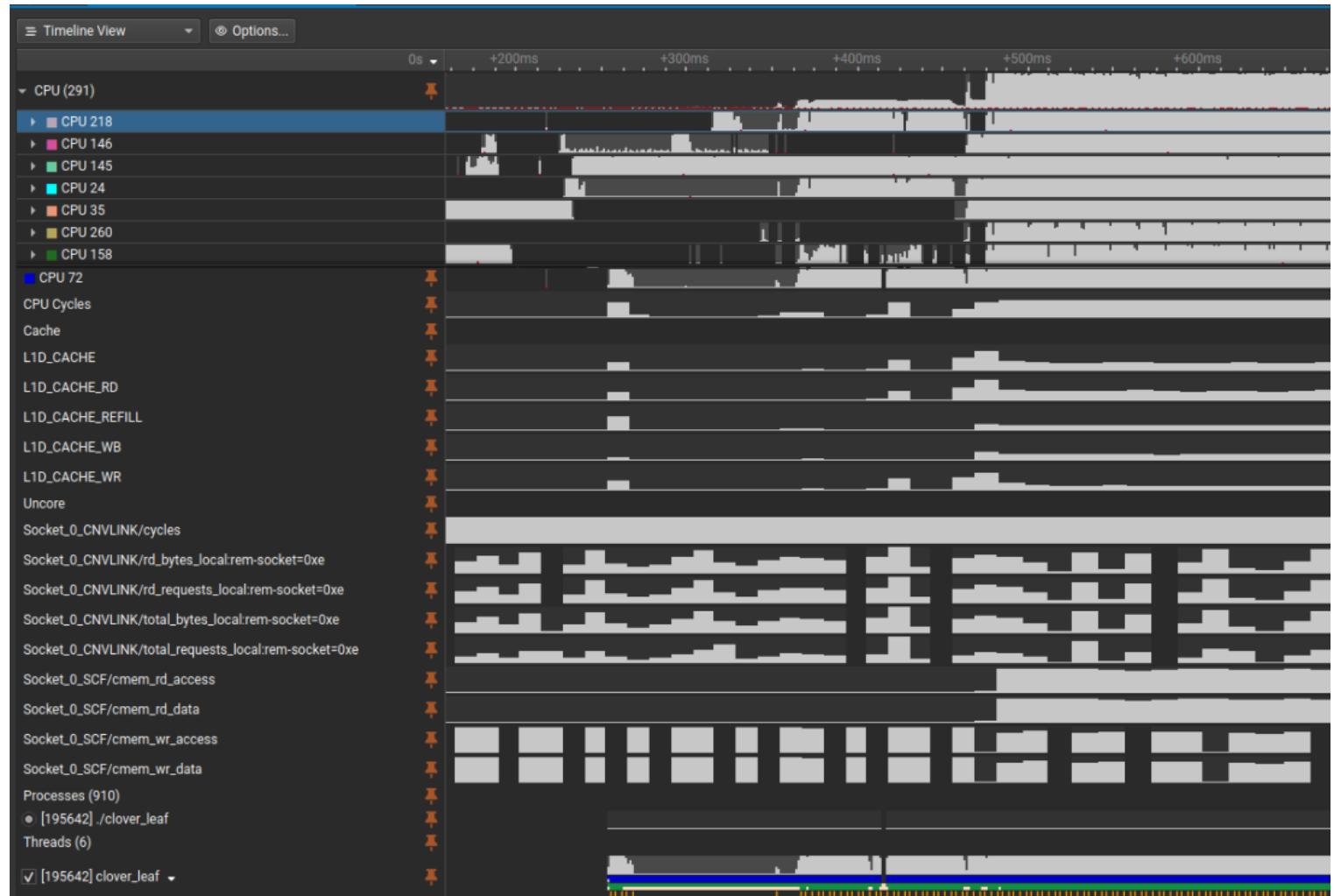




# Grace Host Profiling

## Hardware Counters and Metrics

- CPU Core and Uncore Events
  - Sampled for each CPU
  - Visualize parallelism effects
  - Cache hit/miss, instructions retired, etc...
  - L3 Coherency Fabric
  - Socket to socket traffic
- Variable sampling frequencies supported
- Timeline correlated with all other data
  - GPU vs. CPU idle times and metrics
  - Data movement
  - Zoom and filter

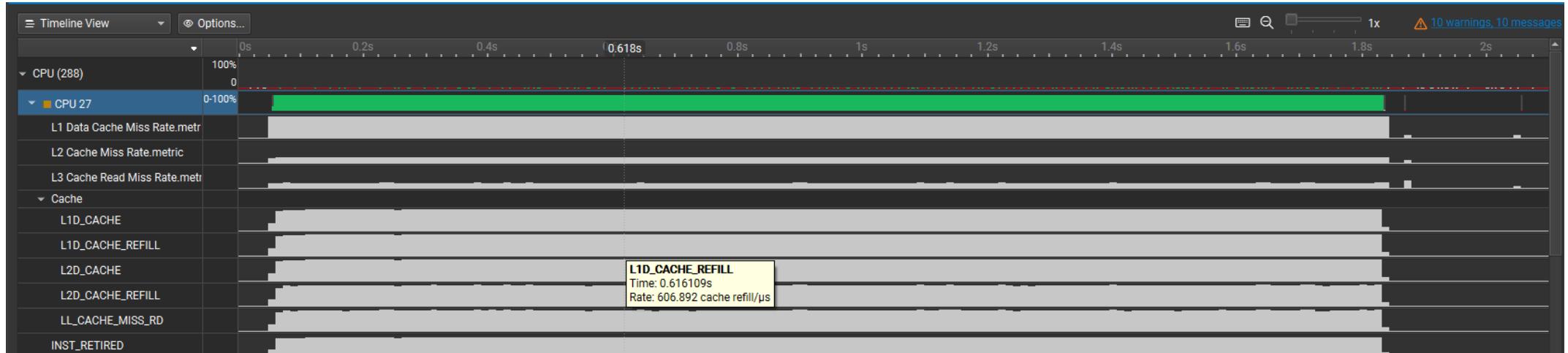




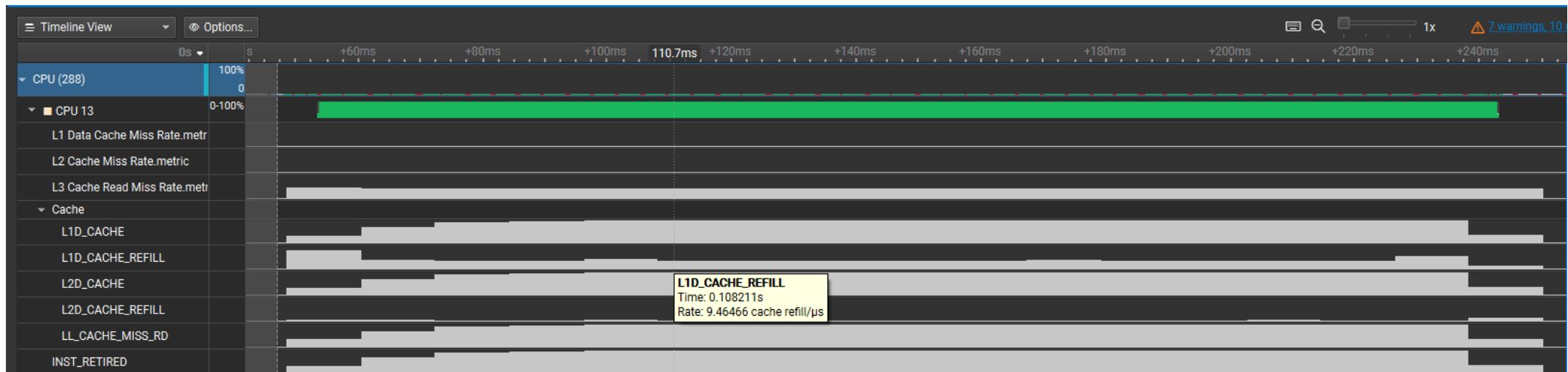
# Grace Host Profiling

## Cache Access Pattern Example

Single threaded CPU matrix multiplication with poor memory access patterns



Improving access pattern and implementing cache blocking





# JupyterLab Integration Updates

- Extension to JupyterLab
- Profile individual Jupyter cells
- Text-based results can be viewed directly in Jupyter
- Launch **new** remote GUI streaming container directly in JupyterLab
  - Servers without X, Windowing Manager, ...
  - Container with X, WM, & WebRTC server
  - Dockerfile inside Nsight Systems Installer
- See it in action:
  - [DLIT61667](#): Profilers, Python, and Performance: Nsight Tools for Optimizing Modern CUDA Workloads

The screenshot shows a JupyterLab interface with a dark theme. The top navigation bar includes File, Edit, View, Run, Kernel, Tabs, NVIDIA Nsight, Settings, and Help. The tabs bar shows 'Numba\_054\_CUDA\_Release.ipynb'. The main content area displays a section titled 'Numba 0.54 CUDA Release Demo' with the following text:

Key changes to the CUDA target for Release 0.54 demonstrated in this notebook:

- Warnings for behavior that may have a negative impact on performance (Michael Collison):
  - Kernel launches using grids that are too small to make effective use of the device
  - Implicit copies that force a costly synchronization when launching a kernel
- Support for implementing warp-aggregated atomics (Graham Markall):
  - The functions `activemask()` and `lanemask_lt()` are now supported
  - `cuda_ffs` now gives correct answers (its behavior mirrors that of `_ffs()` in CUDA C).
- Tuples can now be passed to CUDA ufuncs (Graham Markall).
- Relaxed strides checking is now used to compute contiguity of arrays, enabling some new use cases (Graham Markall).

Other key changes, not demonstrated in this notebook, include:

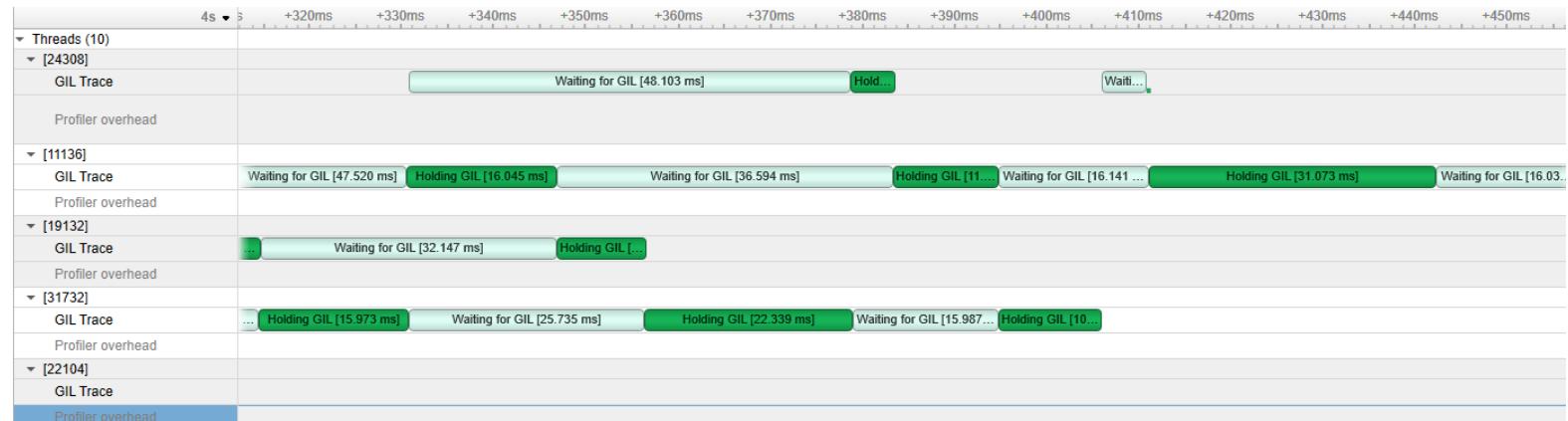
- Debugging improvements (Graham Markall):
  - `cuda-gdb` can now find the source location at the beginning of a kernel (e.g. when breaking on the first instruction of kernels with `set cuda break_on_launch application`)
  - Breakpoints can now be set on mangled names of kernels.
- Per-Thread Default Stream support (Graham Markall)
- Support for adding lineinfo to CUDA kernels so that Nsight Compute can display profile information for each source line of CUDA kernels (Max Katz)

At the bottom, the status bar shows 'Simple' mode, 0 errors, 1 warning, 'Python 3 (ipykernel) | Idle', 'Mode: Command', 'Ln 1, Col 1', 'Numba\_054\_CUDA\_Release\_Demo.ipynb', and 0 notifications.



# Python Profiling Updates

- Python Call Stacks Samples and CUDA API Backtrace
    - Identify where you are and how you got there
  - Global Interpreter Lock (GIL) trace
    - Common performance limiter in Python
  - See annotated code ranges built into in popular frameworks and libraries such as:
    - RAPIDS, Spark, CV-CUDA, and more...



Timeline range for a CUDA API call

## C/C++ frames

## Python frames

```

Call to computeOffsetsKernel
  Kernel launcher
    Begins: 120.838s
    Ends: 120.838s (+14.568 µs)
    Return value: 0
  GPU: 0000:01:00.0 - NVIDIA GeForce GTX 1080
  Stream: 7
  Latency: 13.379 µs→
  Correlation ID: 1318684

Call stack at 120.838s:
  Insight Systems frames
  libcuda.so.635.54.03!0x7f13f39c462e6
  libcuda_cnn_infer.so.8!0x7f3e15a11d58
  libcuda_cnn_infer.so.8!lccask_cudnn::ImplicitGemmShader<...>::initDeviceReservedSpace(...
  libcuda_cnn_infer.so.8!lccudnn::cmn::infer::InferNDSubEngine<...>::execute_internal_fprop_Impl(...
  libcuda_cnn_infer.so.8!lccudnn::cmn::infer::InferNDSubEngine<...>::execute_internal_Impl(...
  libcuda_cnn_infer.so.8!lccudnn::cmn::EngineInterface::execute(...
  libcuda_cnn_infer.so.8!lccudnn::cmn::EngineContainer<...>::execute_internal_Impl(...
  libcuda_cnn_infer.so.8!lccudnn::cmn::EngineInterface::execute(...
  libcuda_cnn_infer.so.8!lccudnn::cmn::AutoTransformationExecutor::execute_pipeline(...
  libcuda_cnn_infer.so.8!lccudnn::cmn::BatchPartitionExecutor::operator[](...) const
  libcuda_cnn_infer.so.8!lccudnn::cmn::GeneralizedConvolutionEngine<...>::execute_internal_Impl(...
  libcuda_cnn_infer.so.8!lccudnn::cmn::EngineInterface::execute(...
  libcuda_cnn_infer.so.8!lccudnn::backend::execute(...
  libcuda_cnn_infer.so.8!lccudnnBackendExecute
  btorch_cuda.solat::native::run_conv_plan(...
  btorch_cuda.solat::native::run_single_conv(...
  btorch_cuda.solat::native::raw_cudnn_convolution_forward_out(...
  btorch_cuda.solat::native::cudnn_convolution_forward(...
  btorch_cuda.solat::native::cudnn_convolution(...
  btorch_cuda.solat::native::wrapper_CUDA_cudnn_convolution(...
  btorch_cuda.solat::impl::wrap_kernel_functor_unboxed<...>::call(...
  btorch_cpu.solat::ops::cudnn_convolution::call(...
  btorch_cpu.solat::native::convolution(...
  btorch_cpu.solat::native::wrapper_CompositeExplicitAutograd__convolution(...
  btorch_cpu.solat::impl::wrap_kernel_functor_unboxed<...>::call(...
  btorch_cpu.solat::ops::convolution::call(...
  btorch_cpu.solat::native::convolution(...
  btorch_cpu.solat::native::wrapper_CompositeExplicitAutograd__convolution(...
  btorch_cpu.solat::impl::wrap_kernel_functor_unboxed<...>::call(...
  btorch_cpu.solat::ops::convolution::redispatch(...
  btorch_cpu.soltorch::autograd::VariableType<...>::convolution(...
  btorch_cpu.soltorch::autograd::VariableType<...>::convolution(...
  btorch_cpu.soltorch::autograd::VariableType<...>::conv2d(...
  btorch_cpu.soltorch::autograd::VariableType<...>::conv2d(...
  btorch_cpu.soltorch::autograd::VariableType<...>::conv2d(...
  btorch_python::torch::autograd::THPVariable_conv2d(...
  Python] conv_py!conv_forward_459
  Python] conv_py!forward_463
  Python] module.py!_call_impl_1501
  Python] main.py!forward_24
  Python] module.py!_call_impl_1501
  Python] main.py!train_42
  Python] main.py!main_136
  Python] main.py<module>_145
  bc.so.6!_lIBC_start_call_main
  bc.so.6!_lIBC_start_main@@GLIBC_2
  python3!0!start

```



# Cluster and Recipe Framework Improvements

- Nsight Systems enhanced support for Kubernetes

- Nsight Systems analysis framework:

- User programmable and predefined recipes to

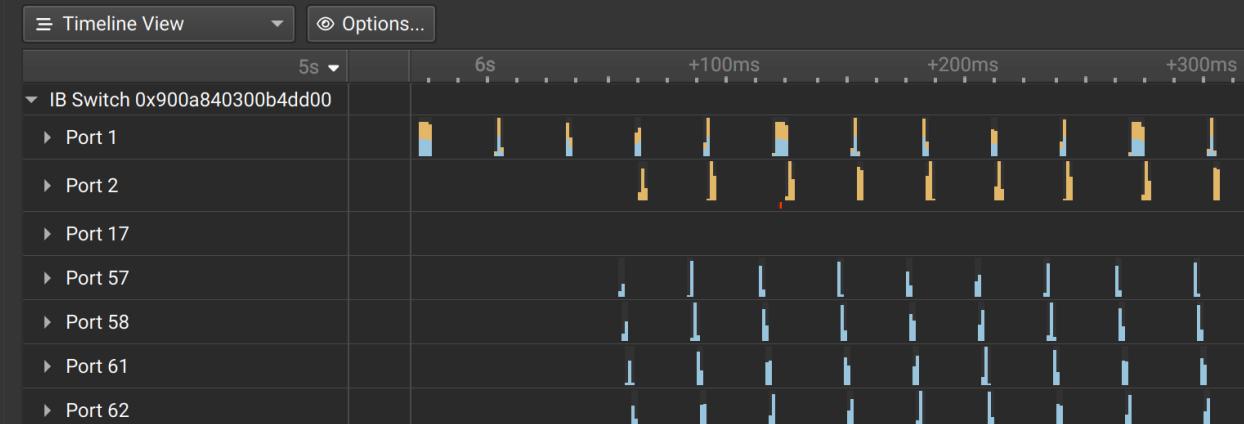
- Process and analyze complex and large reports or collection of reports

- Understand how compute cold-spots relate to communications

- Generate multi-node heatmaps to show :

- InfiniBand congestion
  - InfiniBand, Ethernet, and NVLink throughputs
  - Overlapped compute and networking

- #### 1A. Switch per port support



```

workstation:/develop/Achieve/CSP/devtools-sidecar-injector$ kubectl get pods -A
NAMESPACE     NAME                               READY   STATUS    RESTARTS   AGE
example-ns    cuda-vector-add-69c5cb6b7c-r542t   1/1    Running   0          34s
gmp-system    alertmanager-0                   2/2    Running   0          3d16h
gmp-system    collector-sd8ln                 2/2    Running   0          3d16h
gmp-system    collector-tsjd7                 2/2    Running   0          46m
gmp-system    gmp-operator-69f4b6cb87-lxfk5      1/1    Running   0          46h
gmp-system    rule-evaluator-9bd9c559f-2kzkh     2/2    Running   2 (3d16h ago) 3d16h
gpu-operator  gpu-feature-discovery-lwd45       1/1    Running   0          45m
gpu-operator  gpu-operator-999cc8dcc-cj5hc      1/1    Running   10 (46h ago) 3d15h
gpu-operator  gpu-operator-node-feature-discovery-gc-7cc7ccffff8-2cgbh  1/1    Running   0          3d15h
gpu-operator  gpu-operator-node-feature-discovery-master-d8597d549-l7vj  1/1    Running   0          3d15h
gpu-operator  gpu-operator-node-feature-discovery-worker-hcmr7      1/1    Running   0          46m
gpu-operator  gpu-operator-node-feature-discovery-worker-rvvcz      1/1    Running   9 (46h ago) 3d15h
gpu-operator  nvidia-container-toolkit-daemonset-lqgv7      1/1    Running   0          45m
gpu-operator  nvidia-cuda-validator-k6bph        0/1    Completed  0          41m
gpu-operator  nvidia-dcgm-exporter-29kbz       1/1    Running   0          45m
gpu-operator  nvidia-device-plugin-daemonset-n7rjl    1/1    Running   0          45m
gpu-operator  nvidia-driver-daemonset-jb4dw      1/1    Running   0          45m
gpu-operator  nvidia-operator-validator-56nc4      1/1    Running   0          45m
kube-system   event-exporter-gke-754cff8686-mv585  2/2    Running   0          3d16h
kube-system   fluentbit-gke-d8kg2        2/2    Running   0          46m
kube-system   fluentbit-gke-lrjvb        2/2    Running   0          3d16h
kube-system   gke-metadata-server-8qm55       1/1    Running   0          46m
kube-system   gke-metadata-server-kfcj8        1/1    Running   0          3d16h
kube-system   gke-metrics-agent-nrgcn       2/2    Running   0          46h
kube-system   gke-metrics-agent-x8rc        2/2    Running   0          46m
kube-system   konnectivity-agent-778fc89f85-c96jx  2/2    Running   0          3d15h
kube-system   konnectivity-agent-778fc89f85-stgl5  2/2    Running   0          3d16h
kube-system   konnectivity-agent-autoscaler-8fff66b84-rlz7q  1/1    Running   0          3d16h
kube-system   kube-dns-577947fcfc-2g4x4        4/4    Running   0          3d15h
kube-system   kube-dns-577947fcfc-hrsdh        4/4    Running   0          3d16h
kube-system   kube-dns-autoscaler-755c7fdf5-9b5bv  1/1    Running   0          3d16h
kube-system   kube-proxy-gke-nightsight-load-test-tf-nightsight-load-t-03e52019-1z5m 1/1    Running   0          3d16h
kube-system   kube-proxy-gke-nightsight-load-test-tf-nightsight-load-t-e8d740d6-2jbh  1/1    Running   0          46m
kube-system   17-default-backend-9b4f84c76-wlwnl    1/1    Running   0          3d16h
kube-system   metrics-server-v0.6.3-b76d4c5f8-qvhch  2/2    Running   0          3d16h
kube-system   netd-msn9                     1/1    Running   0          46m
kube-system   netd-vbc92                    1/1    Running   0          3d16h
kube-system   nvidia-gpu-device-plugin-small-ubuntu-54pl6  0/1    Init:0/2  0          46m
kube-system   pdcsi-node-mm2s4                2/2    Running   0          46m
kube-system   pdcsi-node-w8mft                2/2    Running   0          3d16h
nvidia-devtools-sidecar-injector  nvidia-devtools-sidecar-injector-676b496845-jt9rc  1/1    Running   0          46s

workstation:/develop/Achieve/CSP/devtools-sidecar-injector$ # Using nsys_k8s.py we can control the profiling of the containers.
./nsys_k8s.py nsys stop
Executing command: /mnt/nv/bin/nsight-systems/target-linux-x64/nsys stop --session k8s_auto_56b82acc
Output from pod cuda-vector-add-69c5cb6b7c-r542t, container cuda-vector-add:
Generating '/tmp/nsys-report-f50f_qdstrm'
[1/1] [=====100%] auto_sleep_example-ns_cuda-vector-add-69c5cb6b7c-_cuda-vector-add_1780078828373_56b82acc.nsys-report
Generated:
/home/auto_sleep_example-ns_cuda-vector-add-69c5cb6b7c-_cuda-vector-add_1780078828373_56b82acc.nsys-report

-workstation:/develop/Achieve/CSP/devtools-sidecar-injector$
```

# NVIDIA Tools eXtension (NVTX)

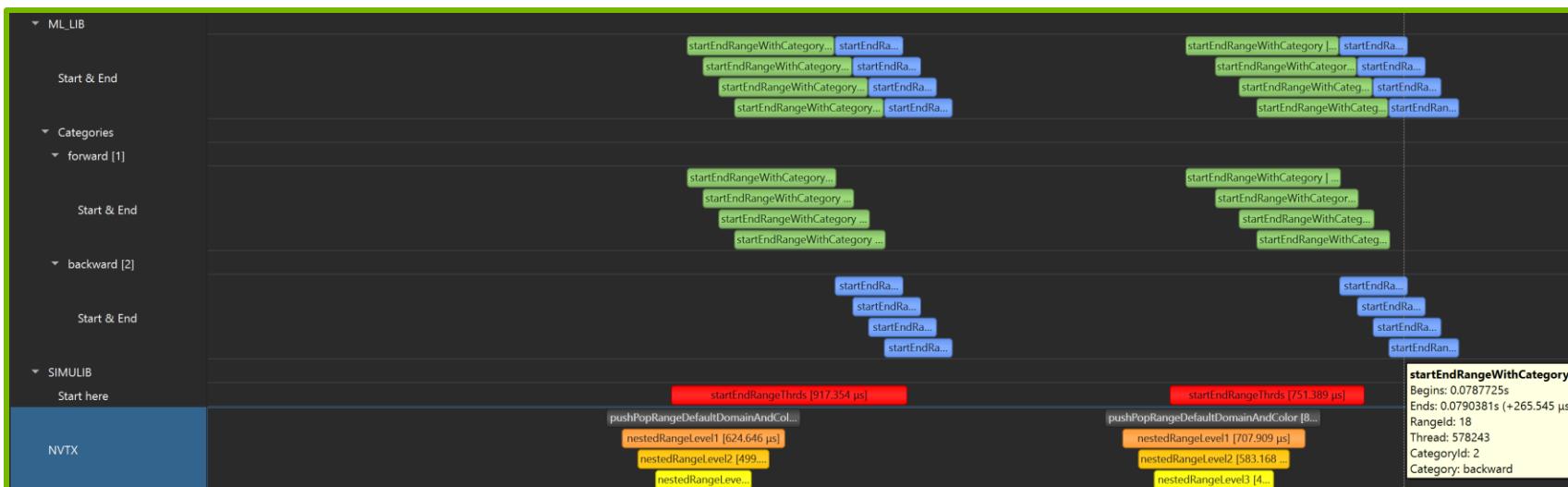
- Decorate application source code with annotations (markers, ranges, nested ranges, ...) to help visualize execution with debugging, tracing and profiling tools
- Header-only library <https://github.com/NVIDIA/NVTX/tree/release-v3/c>.  

```
#include <nvtx3/nvToolsExt.h>
```
- Marker:  

```
nvtxMark("This is a marker");
```
- Push-Pop range  

```
nvtxRangePush("This is a push/pop range");
// Do something interesting in the range
nvtxRangePop(); // Pop must be on same thread as corresponding Push
```
- Start-End range  

```
nvtxRangeHandle_t handle = nvtxRangeStart("This is a start/end range");
// Somewhere else in the code, not necessarily same thread as Start call:
nvtxRangeEnd(handle);
```



API references <https://nvidia.github.io/NVTX/doxygen/index.html> and <https://nvidia.github.io/NVTX/doxygen-cpp/index.html>

# Python and NVTX

- Annotate Python code with NVTX

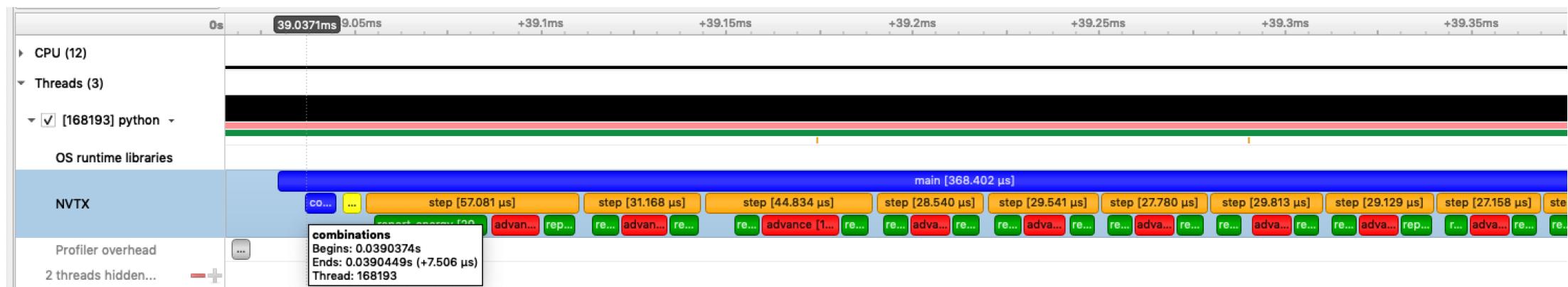
```
# demo.py

import time
import nvtx

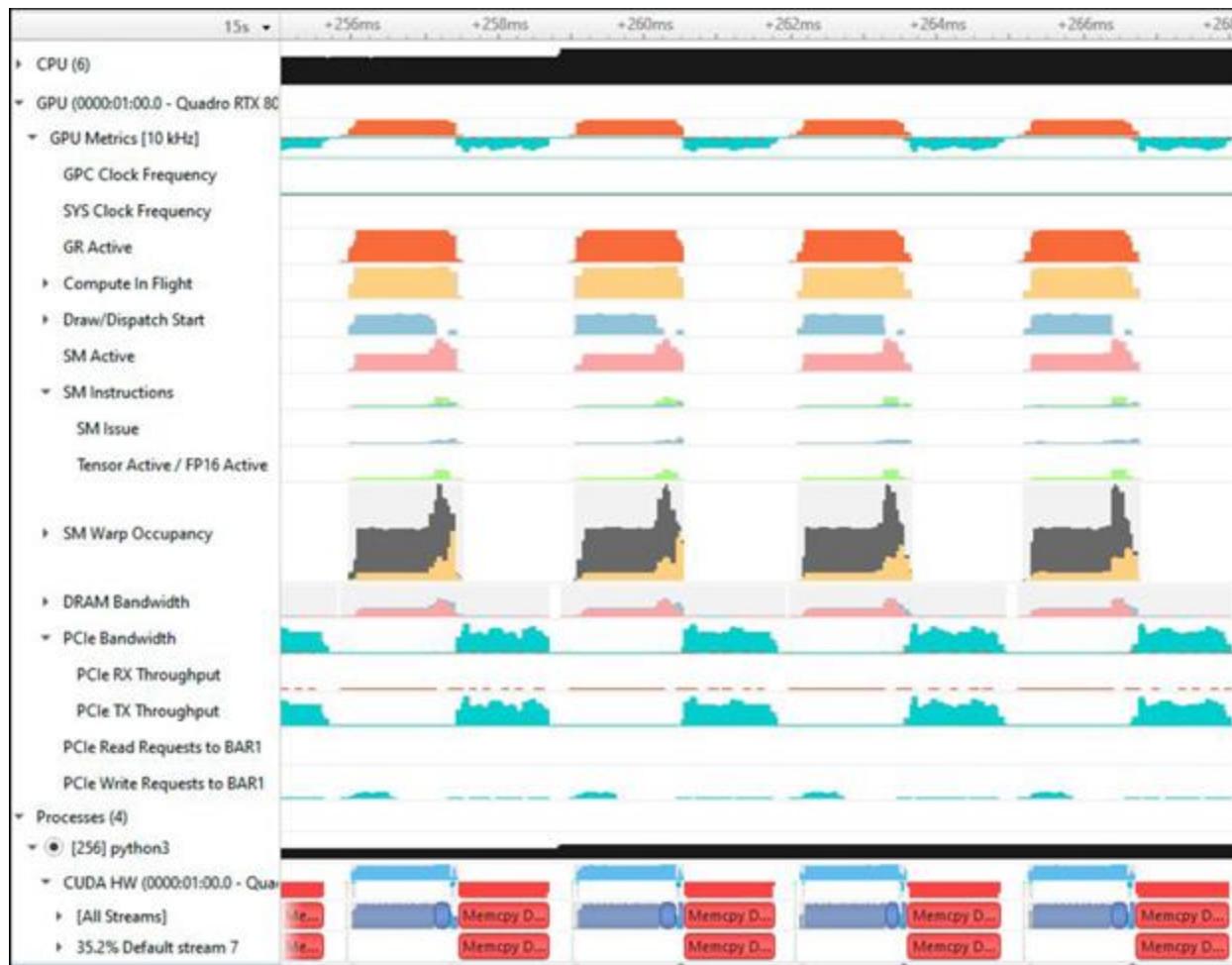
@nvtx.annotate(color="blue")
def my_function():
    for i in range(5):
        with nvtx.annotate("my_loop", color="red"):
            time.sleep(i)

my_function()
```

- pip install nvtx - <https://pypi.org/project/nvtx/>



# GPU Metrics Sampling



- Useful GPU utilization metrics, but no kernel names / correlation

# Interpreting GPU Sampling Metrics

- GR Activity -> GPU is doing work
  - SM, NVENC, NVDEC, Graphics
- SM Activity -> Utilizing width of GPU
  - If low, modify kernel grid dimension or increase batch size
- SM Instruction Issued -> GPU is performing lots of instructions
  - Stalled waiting on memory?
  - Not enough warps to cover memory latency? Issue larger kernel block dimensions.
- SM Instructions tensor activity -> Tensor core utilization
  - Performance up, SM instructions can drop (depending on arch)
  - Can be limited by shared memory, waiting for loads
- Note: Requires disabling **DCGM** and DL built-in profilers

# Application Profiles with Nsight Systems

```
$ nsys profile -o report -stats=true ./myapp.exe
```

- Generated file: report.qdrep (or report.nsys-rep)  
Open for viewing in the Nsight Systems UI
- When using MPI, recommended to use *nsys* after *mpirun/srun*:  

```
$ mpirun -n 4 nsys profile ./myapp.exe
```

# Profiling DL Models

- Pytorch
  - DNN Layer annotations are disabled by default
  - `++ "with torch.autograd.profiler.emit_nvtx():"`
  - Manually with `torch.cuda.nvtx.range_(push/pop)`
  - TensorRT backend is already annotated
- Tensorflow
  - Annotated by default with NVTX in NVIDIA TF containers
  - `TF_DISABLE_NVTX_RANGES=1` to disable for production

# General Optimization Tips

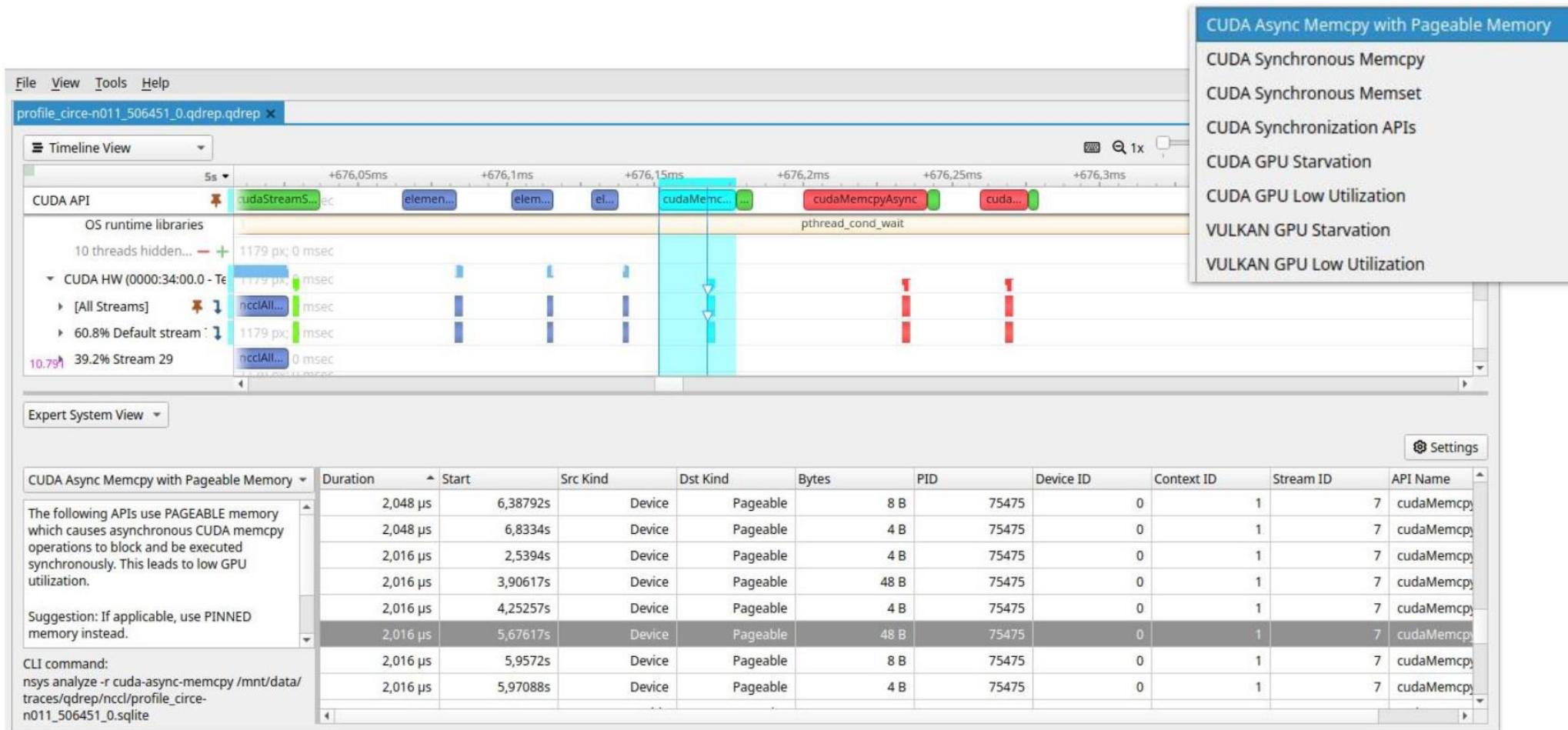
- Using tensor cores?
  - Minimize conversions/transposes
- Increase grid and batch size to utilize GPUs width
- Conventional parallelism – more worker threads!
- Parallel pipelining
  - No data dependency? Parallelize!
  - Prefetch next batch/iteration during computation
- Can I reorder sooner?

# General Optimization Tips

- Fuse tiny kernels, copies, memsets.
  - Check out CUDA Graphs
- Overlap/oversubscribe with MPS
- Multi-buffering
  - Don't make everyone wait on the same piece of memory
  - Double, triple buffer
- Avoid moving data back to the CPU
  - Pre-allocate and recycle!
- Minimize managed memory page faults
  - Prefetch!

# Expert Systems & Statistics

Built-in data analytics with advice





# Nsight Compute

## Kernel Profiler

### Key Features:

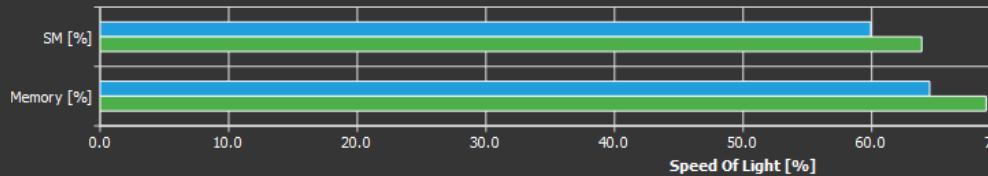
- Interactive CUDA API debugging and kernel profiling
- Built-in rules expertise
- Fully customizable data collection and display
- Command Line, Standalone, IDE Integration, Remote Targets
- OS: Linux (x86, Power, Tegra, Arm SBSA), Windows, macOS X (host only)
- GPUs: Volta+
- Docs/product: <https://developer.nvidia.com/nsight-compute>

### GPU Speed Of Light

High-level overview of the utilization for compute and memory resources of the GPU. For each unit, the Speed Of Light (SOL) reports the achieved percentage of peak performance.

SOL SM [%]	59.93	(-6.20%)	Duration [usecond]
SOL Memory [%]	64.50	(-6.38%)	Elapsed Cycles [cycle]
SOL L1/TEX Cache [%]	26.92	(-5.33%)	SM Active Cycles [cycle]
SOL L2 Cache [%]	64.50	(-6.38%)	SM Frequency [cycle/nsecond]
SOL DRAM [%]	51.55	(+84.34%)	DRAM Frequency [cycle/nsecond]

### GPU Utilization



inst_executed [inst]	63,021,056 (284 instances)
tex_data_bank_conflicts_pipe_lsu_mem_shared_op_ld.sum	0
tex_data_bank_conflicts_pipe_lsu_mem_shared_op_st.sum	0
tex_data_bank_reads.avg.pct_of_peak_sustained_elapsed [%]	9.66
tex_data_bank_writes.avg.pct_of_peak_sustained_elapsed [%]	3.23
tex_data_pipe_lsu_wavefronts.avg.pct_of_peak_sustained_elapsed [%]	46.16
tex_data_pipe_lsu_wavefronts_mem_shared_cmd_read.sum	25,165,824
tex_data_pipe_lsu_wavefronts_mem_shared_cmd_read.sum.pct_of_peak_sustained_active [%]	40.75
tex_data_pipe_lsu_wavefronts_mem_shared_cmd_write.sum	2,097,152
tex_data_pipe_lsu_wavefronts_mem_shared_cmd_write.sum.pct_of_peak_sustained_active [%]	3.40
tex_data_pipe_tex_wavefronts.avg.pct_of_peak_sustained_elapsed [%]	0
tex_f_wavefronts.avg.pct_of_peak_sustained_elapsed [%]	0.00
tex_lsu_writeback_active.avg.pct_of_peak_sustained_elapsed [%]	42.59
tex_lsu_writeback_active.sum [cycle]	27,803,648
tex_lsu_writeback_active.sum.pct_of_peak_sustained_active [%]	45.03
tex_lsu_requests.avg.pct_of_peak_sustained_elapsed [%]	66.00
tex_m_ltex2bar_req_cycles_active.avg.pct_of_peak_sustained_elapsed [%]	3.40
tex_m_ltex2bar_write_bytes.sum [Mbyte]	4.19
tex_m_ltex2bar_write_bytes_mem_global_op_red.sum [byte]	0



# Nsight Compute GUI Interface

Targeted metric sections

Customizable data collection and presentation

Built-in expertise for Guided Analysis and optimization

The screenshot shows the Nsight Compute GUI interface displaying a GPU Speed Of Light report for a softmax compute kernel. The report includes a table of utilization metrics, a bar chart of GPU utilization, and a section for recommendations.

**GPU Speed Of Light**

Metric	Value
SOL SM [%]	45.88
SOL Memory [%]	43.42
SOL TEX [%]	55.37
SOL L2 [%]	13.66
SOL FB [%]	43.42
Duration [usecond]	15.65
Elapsed Cycles [cycle]	16,235
SM Active Cycles [cycle]	12,110.30
SM Frequency [cycle/nsecond]	1.04
Memory Frequency [cycle/usecond]	701.94

**GPU Utilization**

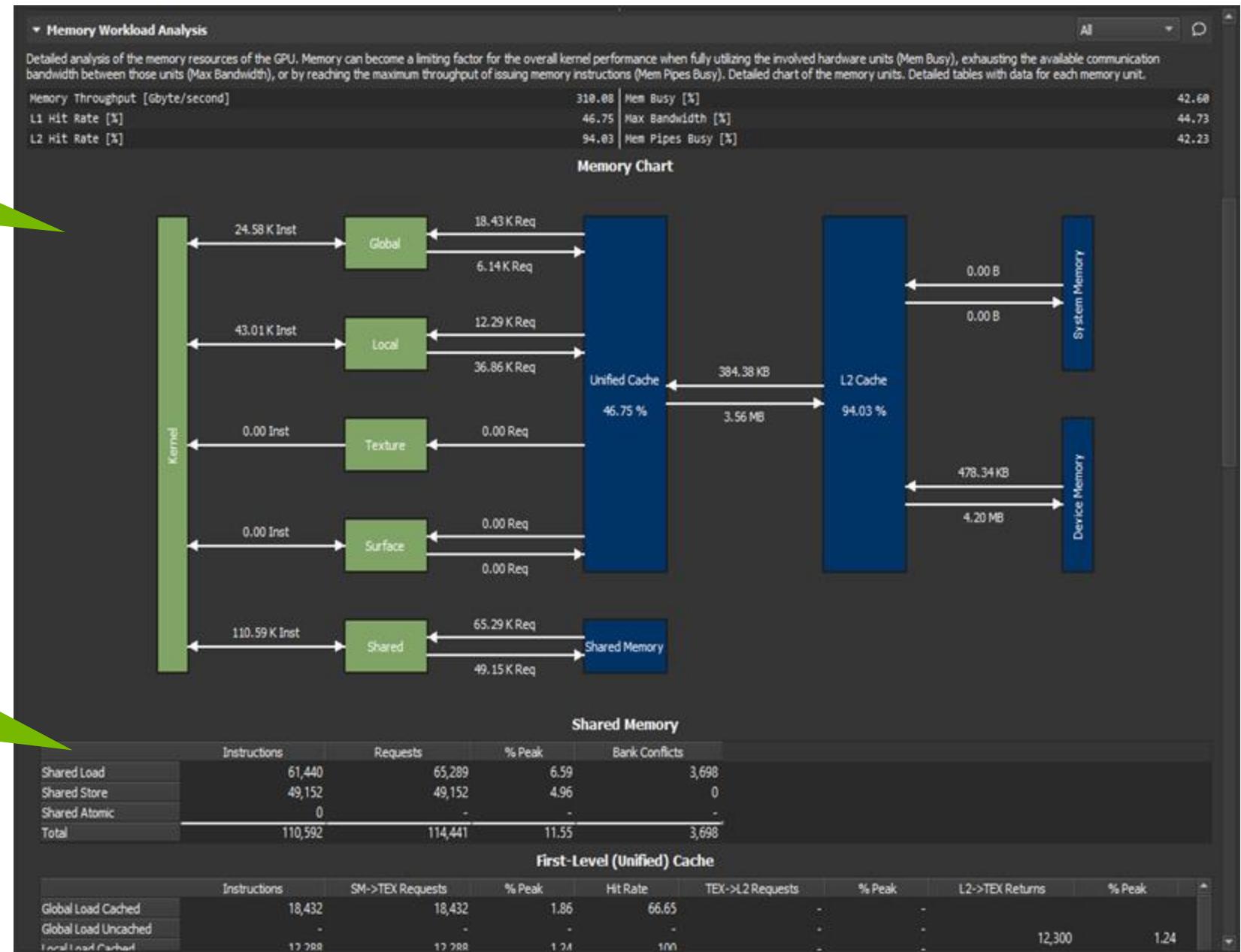
**Recommendations**

**Bottleneck** [Warning] This kernel exhibits low compute throughput and memory bandwidth utilization relative to the peak performance of this device. Achieved compute throughput and/or memory bandwidth below 60.0% of peak typically indicate latency issues. Look at 'Scheduler Statistics' and 'Warp State Statistics' for potential reasons.

**Compute Workload Analysis**

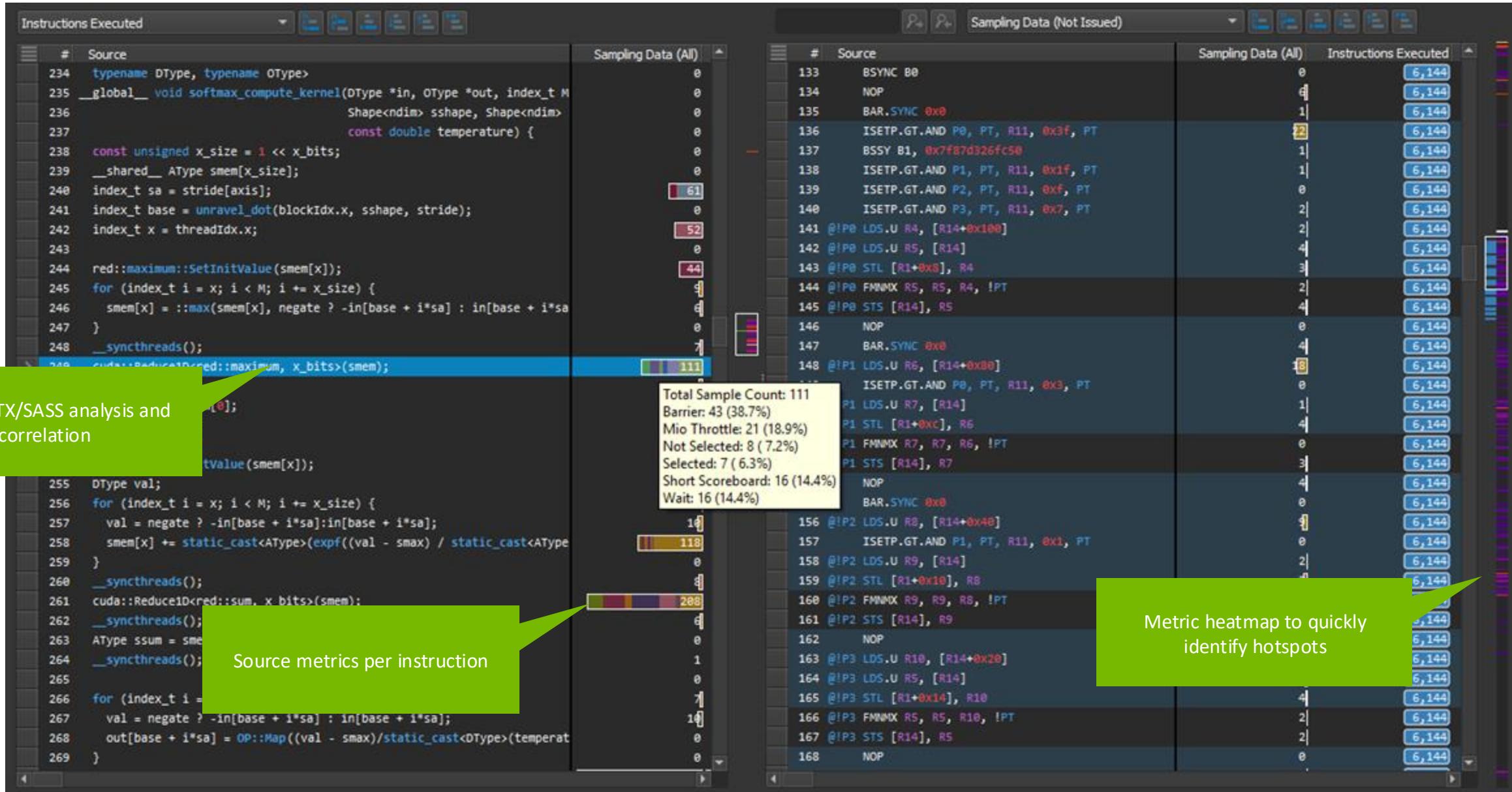
Detailed analysis of the compute resources of the streaming multiprocessors (SM), including the achieved instructions per clock (IPC) and the utilization of each available pipeline. Pipelines with very high utilization might limit the overall performance.

Metric	Value
Executed Ipc Elapsed [inst/cycle]	1.83
Executed Ipc Active [inst/cycle]	2.44
Issued Ipc Active [inst/cycle]	2.46
SM Busy [%]	61.39
Issue Slots Busy [%]	61.39
-	-



Visual memory analysis chart

Metrics for peak performance ratios



## Kernel Profiles with Nsight Compute

```
$ ncu -k mykernel -o report ./myapp.exe
```

- Generated file: report.ncu-report
  - Open for viewing in the Nsight Compute UI
- (Without the –k option, Nsight Compute will profile everything and take a long time)

# Standalone Source Viewer

- View of side-by-side assembly and correlated source code for CUDA kernels
- No profile required
- Open .cubin files directly
- Helps identify compiler optimizations and inefficiencies

The screenshot shows the Standalone Source Viewer interface. At the top, there are tabs for 'Source' and 'Launch'. The 'Launch' tab displays the following information:

Launch	Time	Cycles	Regs	GPU	SM Frequency	CC	Process
32655 - device_tea_leaf_ppcg_solve_update_r (126, 1001, 1)x(32, 4)	1.07 msecound	1,458,003	32	NVIDIA GeForce RTX 2080 Ti	1.36 cycle/nusecond	7.5	[10906] tea_leaf

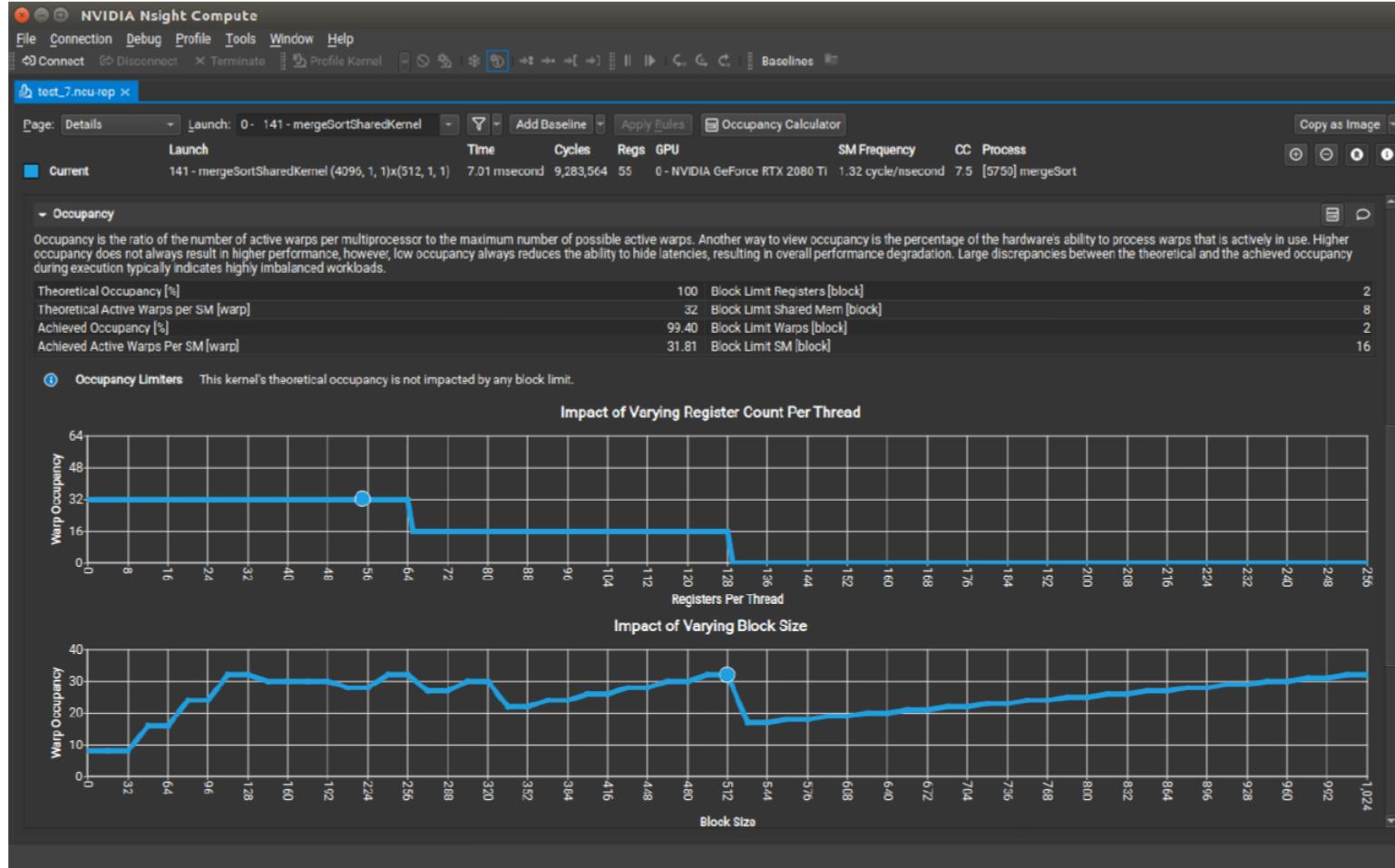
The 'View' dropdown is set to 'Source and SASS'. Below this, there are two panes. The left pane shows the source code for `device_tea_leaf_ppcg_solve_update_r`:#include <tharr2d.h>
#include <math.h>

\_\_global\_\_ void device\_tea\_leaf\_ppcg\_solve\_update\_r(
 kernel\_info\_t kernel\_info,
 double \* \_\_restrict const rtemp,
 const double \* \_\_restrict const Kx,
 const double \* \_\_restrict const Ky,
 const double \* \_\_restrict const sd)
{
 \_\_kernel\_indexes;

 if (WITHIN\_BOUNDS)
 {
 const double result = (1.0
 + (Ky[THARR2D(0, 1, 0)] + Ky[THARR2D(0, 0, 0)])
 + (Kx[THARR2D(1, 0, 0)] + Kx[THARR2D(0, 0, 0)])\*sd[THARR2D(0, 0, 0)]
 - (Ky[THARR2D(0, 1, 0)]\*sd[THARR2D(0, 1, 0)] + Ky[THARR2D(0, 0, 0)]\*sd[THARR2D(0, -1, 0)])
 - (Kx[THARR2D(1, 0, 0)]\*sd[THARR2D(1, 0, 0)] + Kx[THARR2D(0, 0, 0)]\*sd[THARR2D(-1, 0, 0)]));
 }
}The right pane shows the corresponding assembly instructions:22 00007f72 42fa6a50 IADD3 R5, R3, 0x1, R2
23 00007f72 42fa6a60 MOV R28, 8x8
24 00007f72 42fa6a70 IMAD R21, R3, R2, R0
25 00007f72 42fa6a80 IMAD R5, R2, R5, R0
26 00007f72 42fa6a90 IMAD R3, R3, R2, -R2
27 00007f72 42fa6aa0 IMAD.WIDE R16, R21, R20, c[0x0][0x1a0]
28 00007f72 42fa6ab0 IADD3 R3, R0, R3, R2
29 00007f72 42fa6ac0 IMAD.WIDE R10, R5, R20, c[0x0][0x1a0]
30 00007f72 42fa6ad0 IMAD.WIDE R18, R3, R20, c[0x0][0x1a0]
31 00007f72 42fa6ae0 LDG.E.64.CONSTANT.SYS R16, [R16]
32 00007f72 42fa6af0 IMAD.WIDE R26, R21, R20, c[0x0][0x190]
33 00007f72 42fa6b00 LDG.E.64.CONSTANT.SYS R10, [R10]
34 00007f72 42fa6b10 IMAD.WIDE R28, R21, R20, c[0x0][0x1a0]
35 00007f72 42fa6b20 LDG.E.64.CONSTANT.SYS R18, [R18]
36 00007f72 42fa6b30 IMAD.WIDE R12, R5, R20, c[0x0][0x1a0]
37 00007f72 42fa6b40 LDG.E.64.CONSTANT.SYS R14, [R26]
38 00007f72 42fa6b50 LDG.E.64.CONSTANT.SYS R6, [R26+0x8]
39 00007f72 42fa6b60 LDG.E.64.CONSTANT.SYS R2, [R28+0x8]
40 00007f72 42fa6b70 LDG.E.64.CONSTANT.SYS R12, [R12]
41 00007f72 42fa6b80 LDG.E.64.CONSTANT.SYS R8, [R28+0x8]
42 00007f72 42fa6b90 LDG.E.64.CONSTANT.SYS R4, [R28]
43 00007f72 42fa6ba0 IMAD.WIDE R20, R21, R20, c[0x0][0x190]
44 00007f72 42fa6bb0 LDG.E.64.SYS R22, [R20]
45 00007f72 42fa6be0 DADD R24, R16, R10
46 00007f72 42fa6bd0 DMUL R18, R16, R18

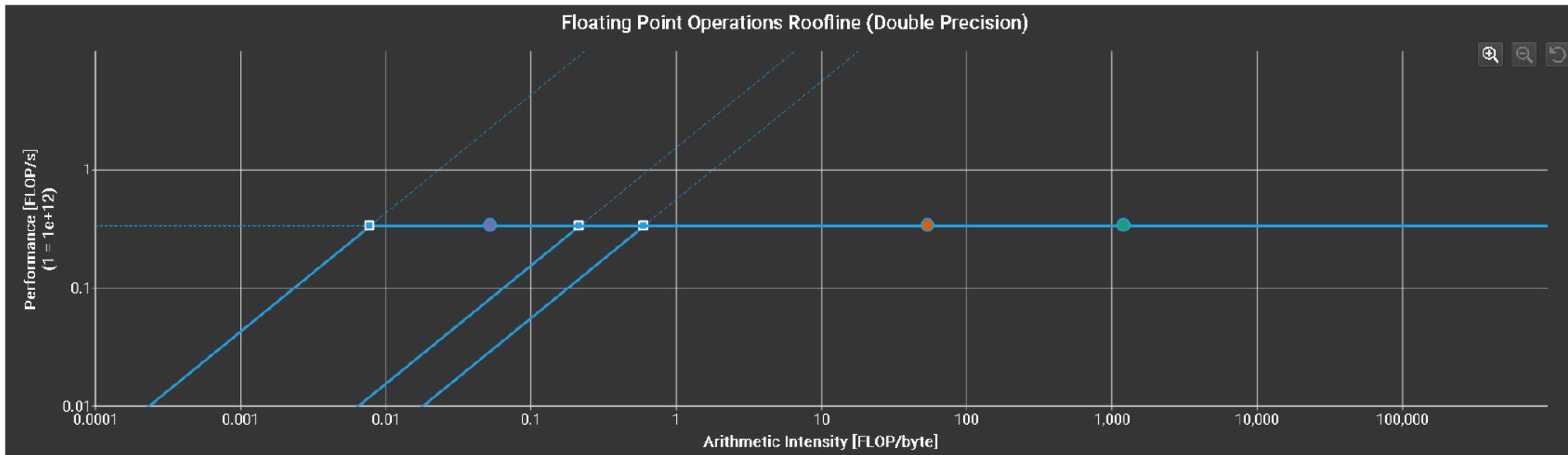
# Occupancy Calculator

Model hardware usage and identify limiters



- Model theoretical hardware usage
- Understand limitations from hardware vs. kernel parameters
- Configure model to vary HW and kernel parameters
- Opened from an existing report or as a new activity

# Hierarchical Roofline



- Visualize multiple levels of the memory hierarchy
- Identify bottlenecks caused by memory limitations
- Determine how modifying algorithms may (or may not) impact performance

Sections/Rules Info

Sections/Rules Reload  Enable All  Disable All  Restore

Enter filter

Name	Priority	Description
GPU Speed Of Light Throughput (1)	10	High-level overview of the throughput for computation...
GPU Speed Of Light Roofline Chart (1)	11	High-level overview of the utilization for computation...
GPU Speed Of Light Hierarchical Roofline Chart (Double Precision)	12	High-level overview of the utilization for computation...
GPU Speed Of Light Hierarchical Roofline Chart (Half Precision)	12	High-level overview of the utilization for computation...
GPU Speed Of Light Hierarchical Roofline Chart (Single Precision)	12	High-level overview of the utilization for computation...
GPU Speed Of Light Hierarchical Roofline Chart (Tensor Core)	12	High-level overview of the utilization for computation...
Compute Workload Analysis (2)	20	Detailed analysis of the compute resources of the system...

# Developer Tools

**Debuggers:** cuda-gdb, Nsight Visual Studio Edition

A screenshot of Microsoft Visual Studio's Debugging interface. The title bar says "matrixMul\_vs2017 (Debugging) - Microsoft Visual Studio". The menu bar includes File, Edit, View, Project, Build, Debug, Team, Nsight, Tools, Test, Analyze, Window, Help. The toolbar has icons for Stop, Start, Break, Continue, and Step Into. The status bar shows "Process: [20680] matrixMul.exe" and "Thread: [1] CUDA Thread". The main window shows the code for "matrixMul.cu" in the Global Scope. A stack frame for "MatrixMulCUDA<int=32>(float\*, float\*, float\*, int, int)" is selected. The Disassembly tab is open, showing assembly code with addresses like 0x7d610228, 0x7d610230, 0x7d610238, 0x7d610240, 0x7d610248, 0x7d610250, 0x7d610258, 0x7d610260, 0x7d610268, 0x7d610270. The assembly code includes instructions like mov, add, and mul.

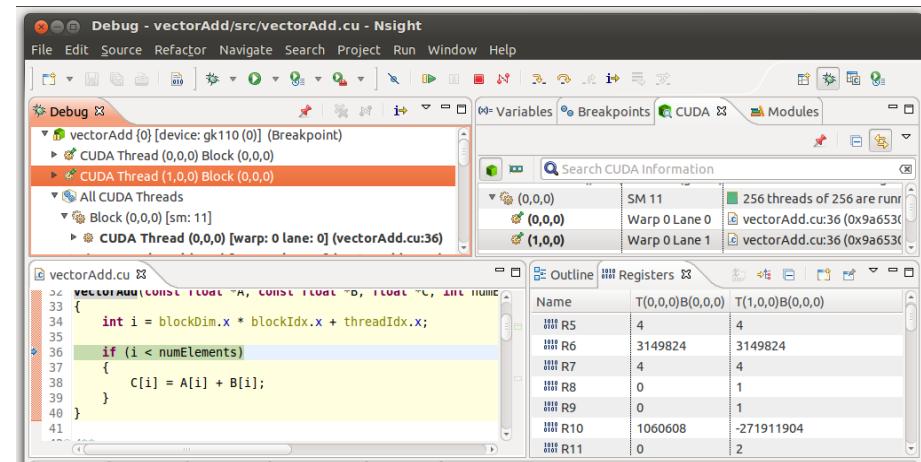
**Profilers:** Nsight Systems, Nsight Compute, CUPTI, NVIDIA Tools eXtension (NVTX)



**Correctness Checker:** Compute Sanitizer

```
$ compute-sanitizer --leak-check full memcheck_demo
===== COMPUTE-SANITIZER
Allocating 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
```

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



# Compute Debuggers

Debug GPU kernels running on device

## ■ CUDA GDB

- CPU + GPU CUDA kernel debugger
- Supports stepping, breakpoints, in-line functions, variable inspection etc...
- Built on GDB and uses many of the same CLI commands
- Local/Remote connection support

## ■ Nsight Visual Studio Edition

- IDE integration for Visual Studio
- Build and Debug CPU+GPU code from Visual Studio

## ■ Nsight Visual Studio Code Edition

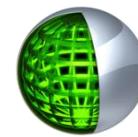
- New IDE integration for VS Code
- Build and Debug CPU+GPU code from Visual Studio Code
- Remotely target Linux targets from Windows or Linux

## ■ Nsight Eclipse Edition

- IDE integration for Eclipse
- Build and Debug CPU+GPU code from Eclipse

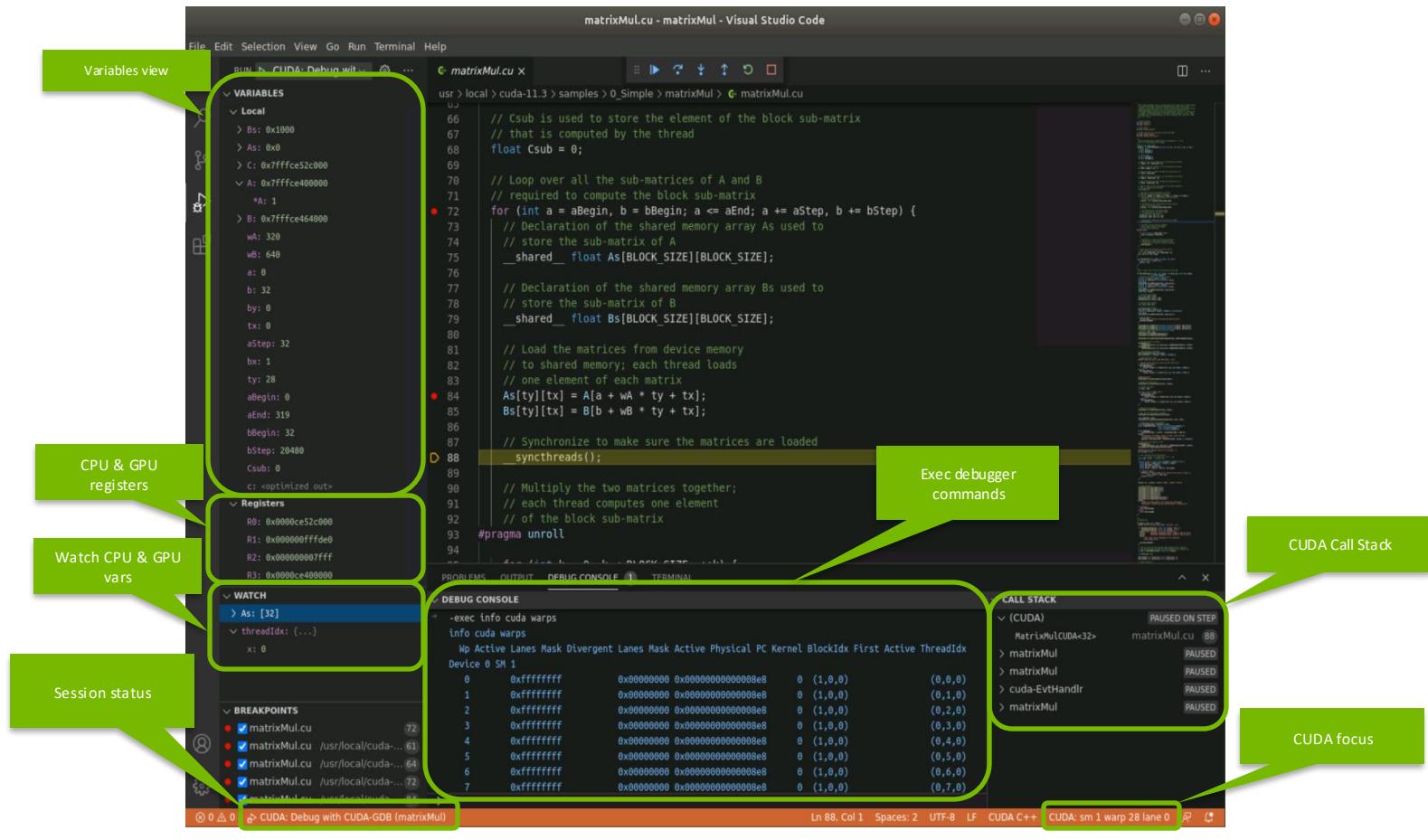
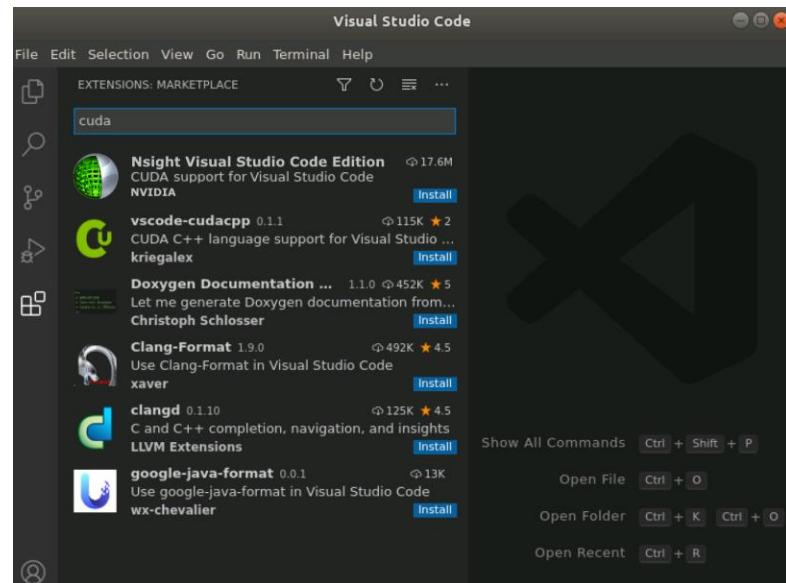
The screenshot shows the Nsight Visual Studio Code Edition interface. On the left, the code editor displays CUDA C code for matrix multiplication. The code includes declarations for shared memory arrays `A` and `B`, and a shared memory array `Csub`. It contains loops for iterating over sub-matrices and calculating element values. On the right, the interface features several panes: a Variables pane showing memory addresses and values for `As`, `Bs`, `Csub`, and `threadIdx`; a Registers pane showing CPU register values; a Watch pane where the value of `As` is being monitored; and a DEBUG CONSOLE pane showing CUDA assembly code and memory access details. The assembly code includes instructions like `MOV R12, R12` and `MOV R11, R11`, along with memory load and store operations.

# Nsight Visual Studio Code Edition



Visual Studio Code extensions that provides:

- CUDA code syntax highlighting
- CUDA code completion
- Build warning/errors
- Debug CPU & GPU code
- Remote connection support via SSH
- Available on the VS Code Marketplace now!



<https://developer.nvidia.com/nsight-visual-studio-code-edition>



# Compute Sanitizer

Automatically Scan for Bugs and Memory Issues

- Compute Sanitizer checks correctness issues via sub-tools:
  - **Memcheck** – Memory access error and leak detection tool.
  - **Racecheck** – Shared memory data access hazard detection tool.
  - **Initcheck** – Uninitialized device global memory access detection tool.
  - **Synccheck** – Thread synchronization hazard detection tool.

<https://github.com/NVIDIA/compute-sanitizer-samples>

```
$ make run_memcheck
/usr/local/cuda/compute-sanitizer/compute-sanitizer --destroy-on-device-error kernel memcheck_demo
=====
===== COMPUTE-SANITIZER
Mallocing memory
===== Invalid __global__ write of size 4 bytes
=====      at 0x70 in unaligned_kernel()
=====      by thread (0,0,0) in block (0,0,0)
=====      Address 0x7f671ac00001 is misaligned
=====      and is inside the nearest allocation at 0x7fb654c00000 of size 4 bytes
=====      Saved host backtrace up to driver entry point at kernel launch time
=====      Host Frame: [0x2774ec]
=====          in /lib/x86_64-linux-gnu/libcuda.so.1
=====      Host Frame: __cudart803 [0xfcdb]
=====          in /home/cuda/github/compute-sanitizer-samples/Memcheck/memcheck_demo
=====      Host Frame: cudaLaunchKernel [0x6a578]
=====          in /home/cuda/github/compute-sanitizer-samples/Memcheck/memcheck_demo
=====      Host Frame: cudaError cudaLaunchKernel<char>(char const*, dim3, dim3, void**, unsigned
=====          in /home/cuda/github/compute-sanitizer-samples/Memcheck/memcheck_demo
=====      Host Frame: __device_stub_Z16unaligned_kernelv() [0xb22e]
=====          in /home/cuda/github/compute-sanitizer-samples/Memcheck/memcheck_demo
=====      Host Frame: unaligned_kernel() [0xb28c]
=====          in /home/cuda/github/compute-sanitizer-samples/Memcheck/memcheck_demo
=====      Host Frame: run_unaligned() [0xaf55]
=====          in /home/cuda/github/compute-sanitizer-samples/Memcheck/memcheck_demo
=====      Host Frame: main [0xb0e2]
=====          in /home/cuda/github/compute-sanitizer-samples/Memcheck/memcheck_demo
=====      Host Frame: .../sysdeps/nptl/libc_start_call_main.h:58:__libc_start_main [0x2df0]
=====          in /lib/x86_64-linux-gnu/libc.so.6
```

# Compute Sanitizer

## Reading a Memcheck Example Report

Address space	Type of access	Access size	Access location
=====	Invalid __global__ write of size 4 bytes		
=====	at 0xb0	Faulty thread	sub/compute-sanitizer-samples/Memcheck/memcheck_demo.cu:39:out_of_bounds_function()
=====	by thread	Faulty address and nearest allocation	
=====	Address 0x87654320 is out of bounds		
=====		bytes before the nearest allocation at 0x7f953da00000 of size 1,024 bytes	Device and host backtraces
=====	Device Frame:	/home/cuda/github/compute-sanitizer-samples/Memcheck/memcheck_demo.cu:44:out_of_bounds_kernel() [0x30]	
=====	Saved host backtrace up to driver entry point at kernel launch time		
=====	Host Frame: [0x2774ec]		
=====		in /lib/x86_64-linux-gnu/libcuda.so.1	
=====	Host Frame: __cudart803 [0xfcdb]		
=====		in /home/cuda/github/compute-sanitizer-samples/Memcheck/memcheck_demo	
=====	Host Frame: cudaLaunchKernel [0x6a578]		
=====		in /home/cuda/github/compute-sanitizer-samples/Memcheck/memcheck_demo	

## Useful Links

Web: <https://developer.nvidia.com/tools-overview>

How to contact us?

Forums: <https://forums.developer.nvidia.com/c/development-tools>

email: [devtools-support@nvidia.com](mailto:devtools-support@nvidia.com)

Other digital GTC talks of interest:

[S21351](#): Scaling the Transformer Model Implementation in PyTorch Across Multiple Nodes

[S21547](#): Rebalancing the Load:Profile-Guided Optimization of the NAMD Molecular Dynamics Program for Modern GPUs using Nsight Systems

S21771: Optimizing CUDA Kernels in HPC Simulation and Visualization Codes using Nsight Compute

[S21565](#): Roofline Performance Model for HPC and Deep-Learning Applications

