

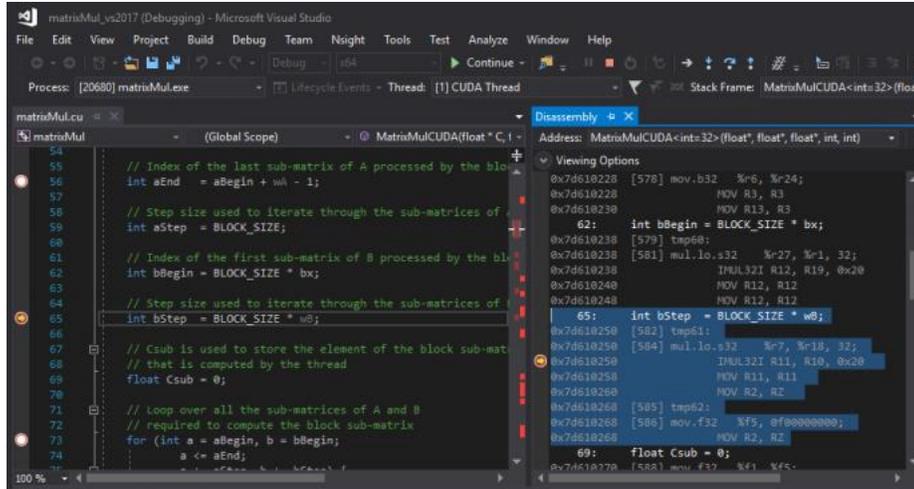
ARGONNE  
**ATPESC10**  
EXTREME-SCALE **COMPUTING**

# NVIDIA Developer Tools

**Kristopher Keipert**  
Solutions Architect, NVIDIA

# DEVELOPER TOOLS

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



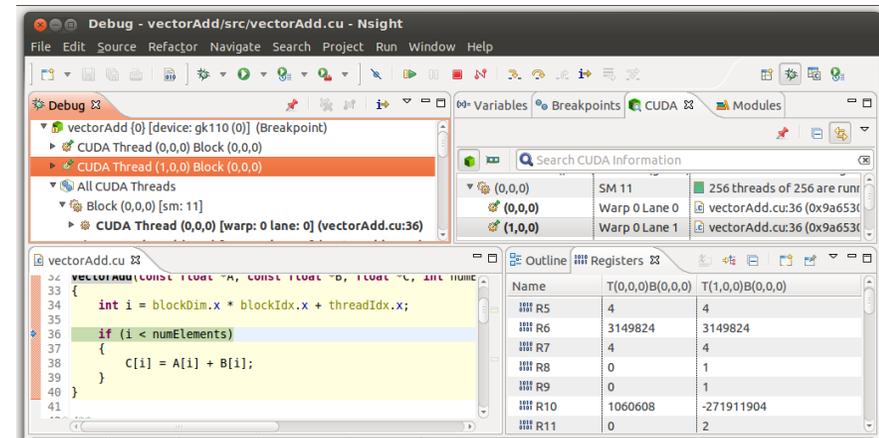
**Profilers:** Nsight Systems, Nsight Compute, CUPTI, NVIDIA Tools eXtension (NVTX)



**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_kernl i(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



# 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

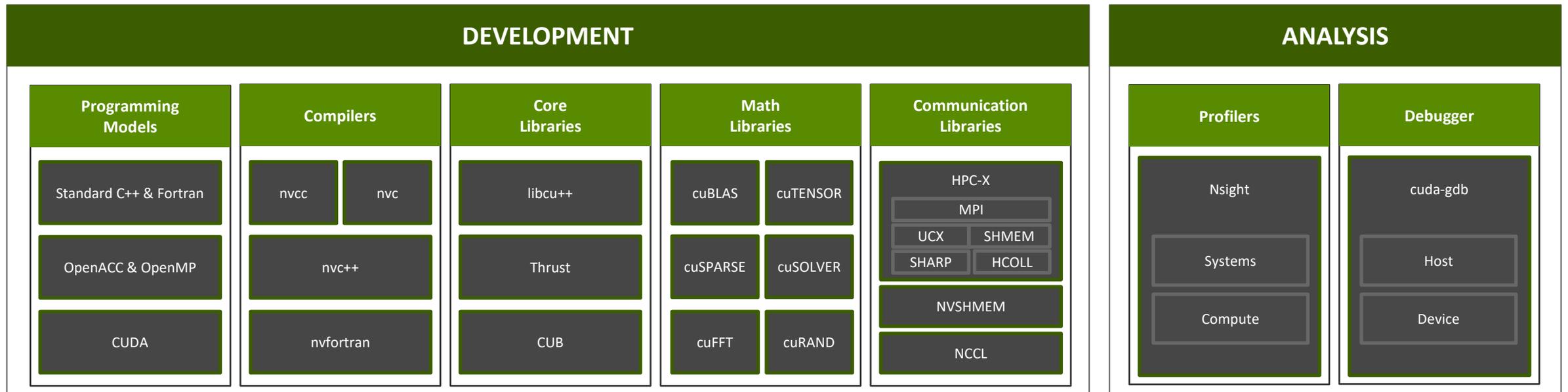
Data Analytics

AI

Quantum

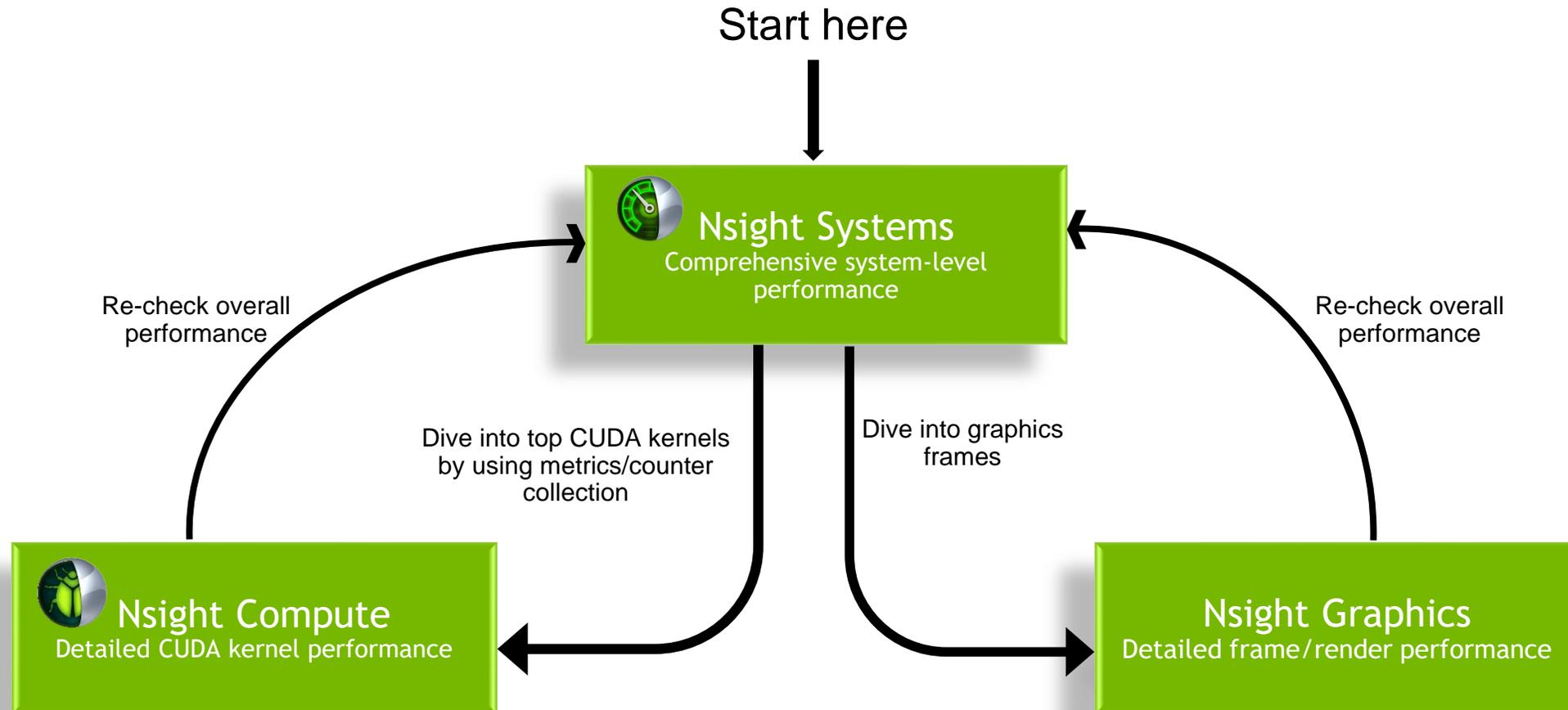
# NVIDIA HPC SDK

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 TOOLS WORKFLOW





# NSIGHT SYSTEMS

## SYSTEM PROFILER

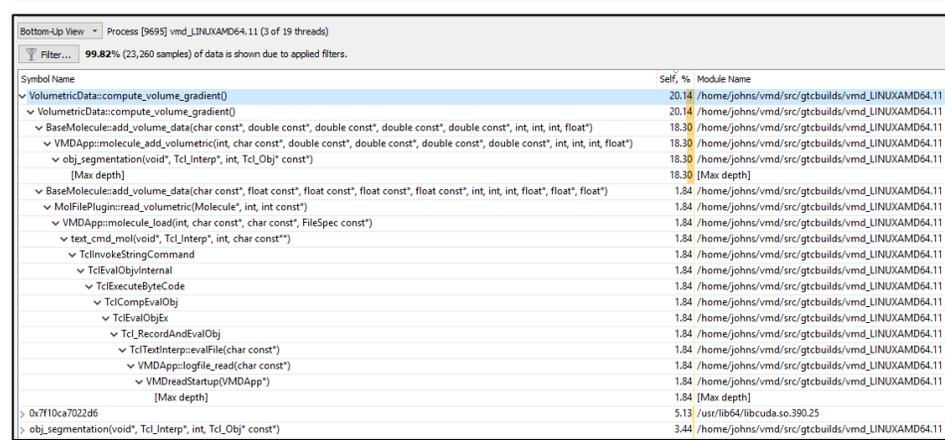
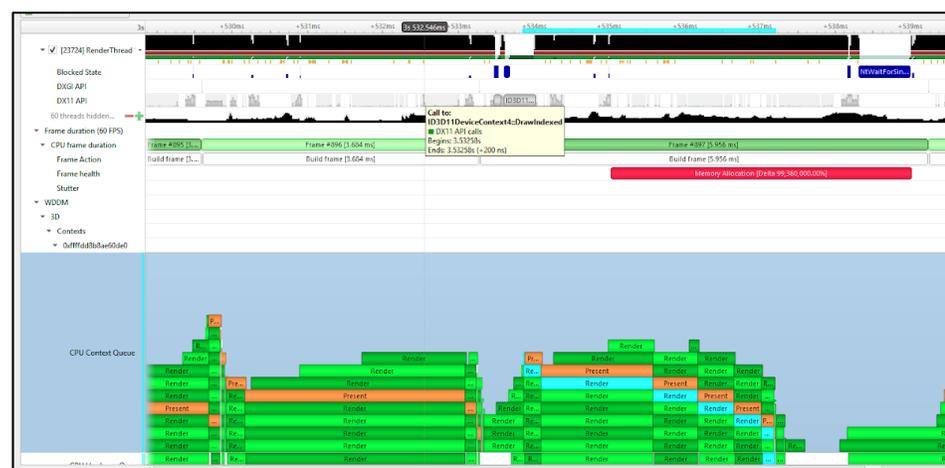
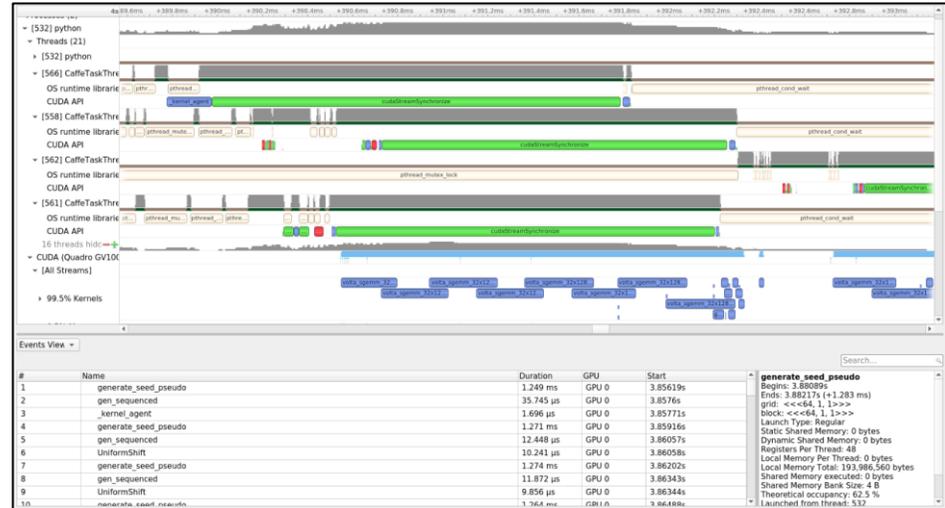
### Key Features:

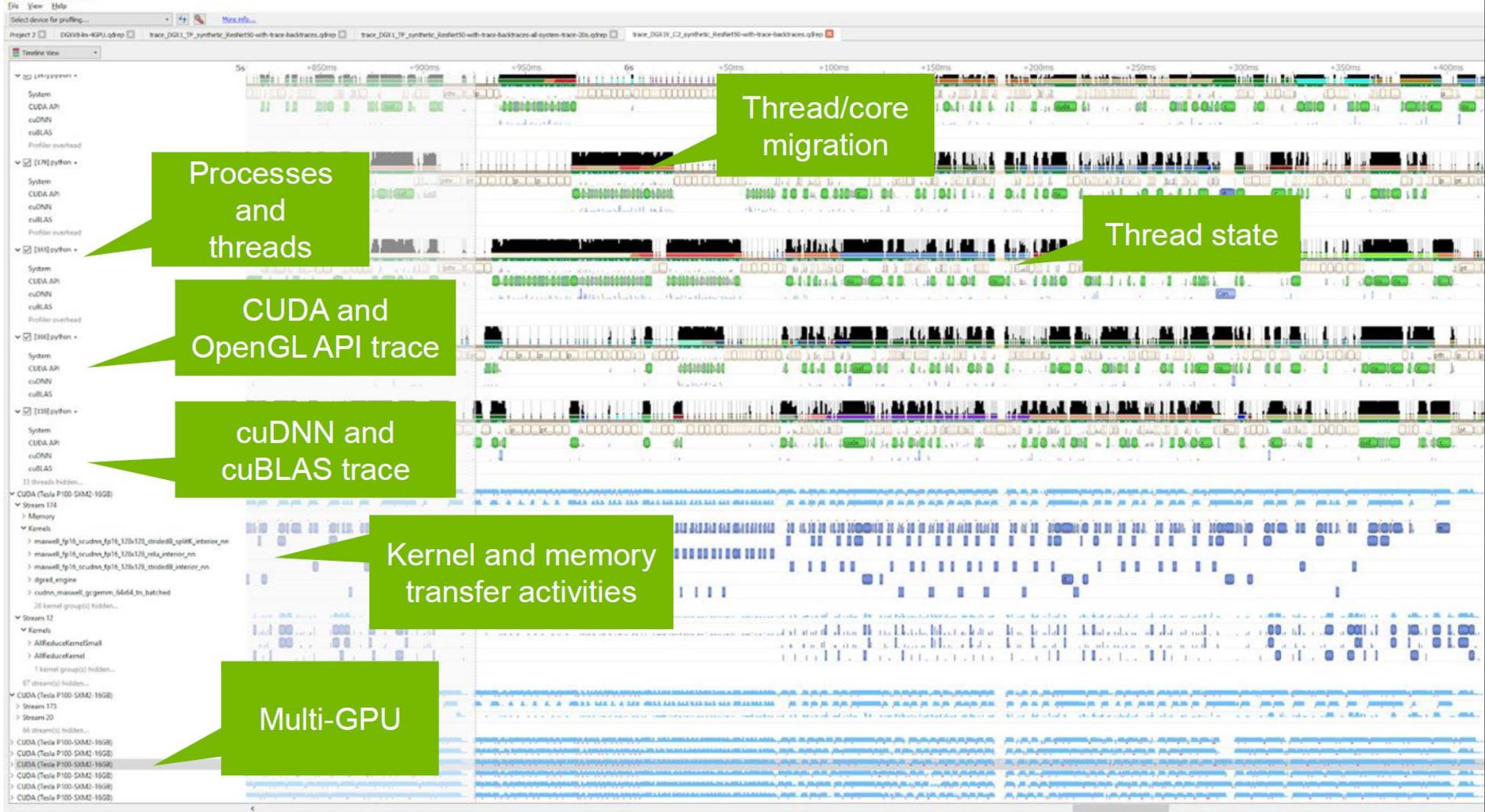
- System-wide application algorithm tuning
  - Multi-process tree support
- Locate optimization opportunities
  - Visualize millions of events on a very fast GUI timeline
  - Or gaps of unused CPU and GPU time
- Balance your workload across multiple CPUs and GPUs
  - CPU algorithms, utilization and thread state
  - GPU streams, kernels, memory transfers, etc
- Command Line, Standalone, IDE Integration

OS: Linux (x86, Power, Arm SBSA, Tegra), Windows, MacOSX (host)

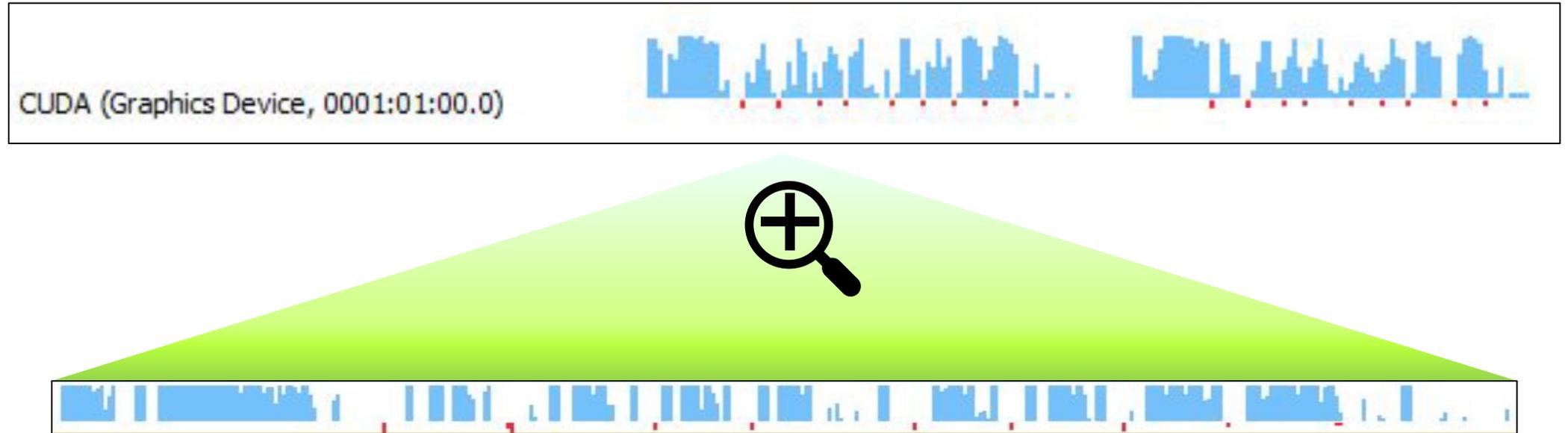
GPUs: Pascal+

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



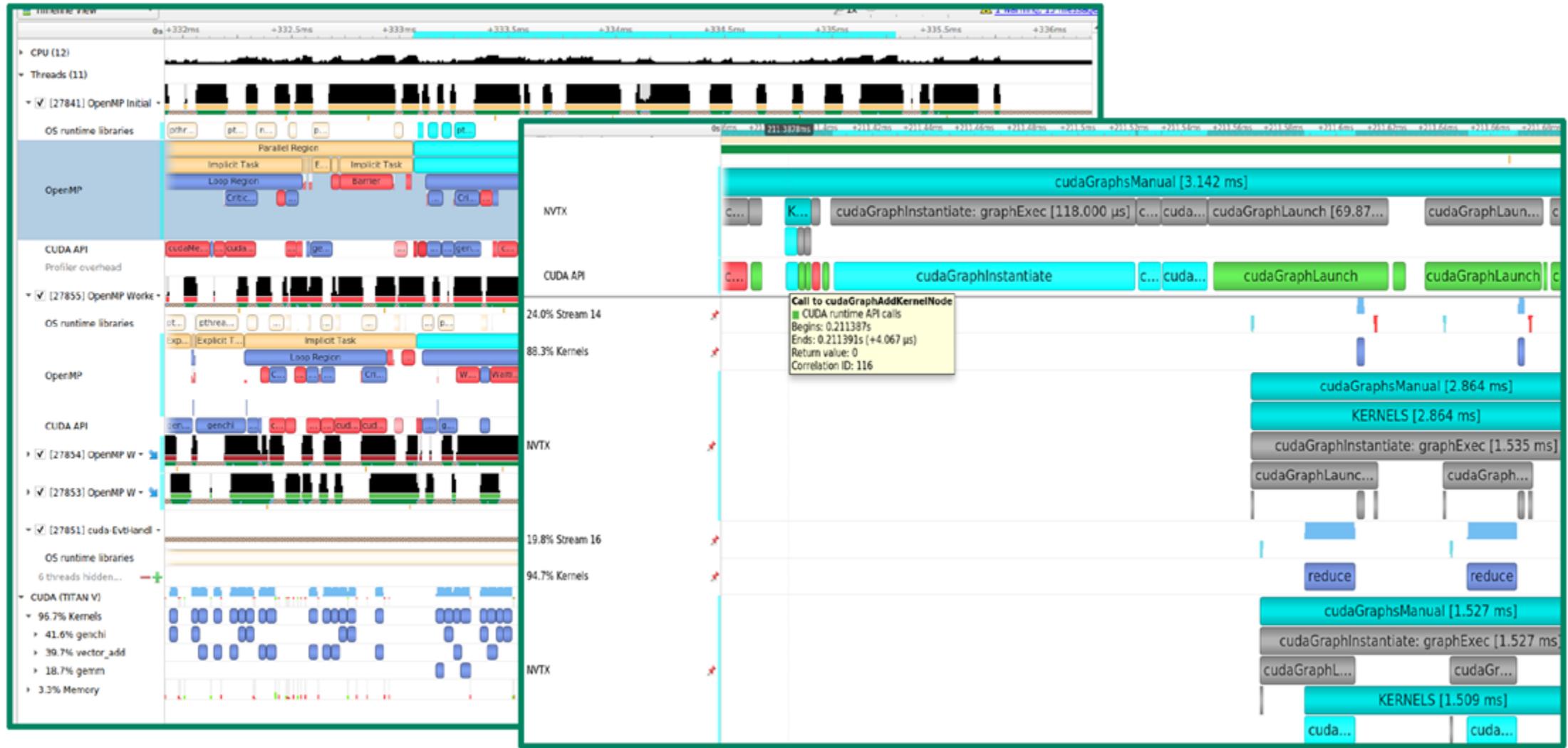


# READING UTILIZATION



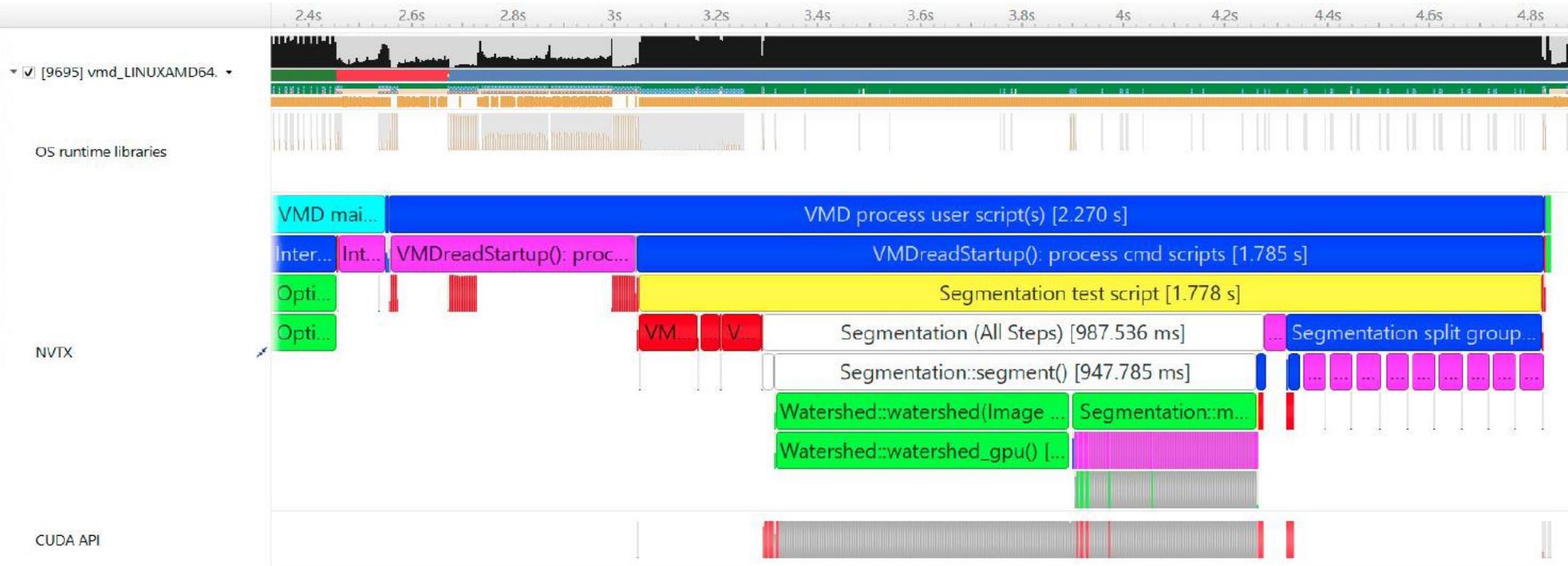
- Zoom in to valleys to find gaps!

# ZOOM/FILTER TO EXACT AREAS OF INTEREST

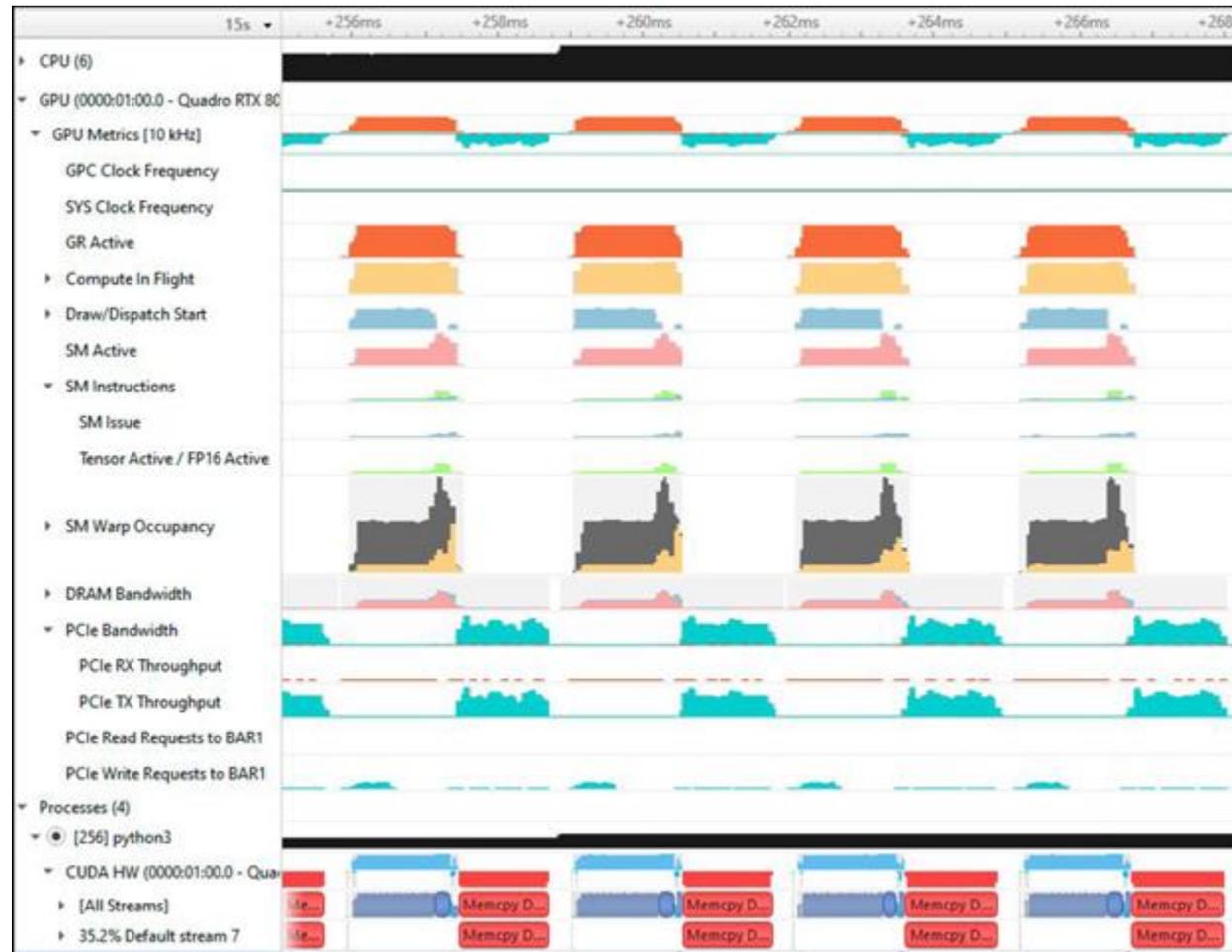


# NVTX: NVIDIA TOOLS EXTENSIONS

## Code Annotation API



# 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

# QUESTIONS TO GUIDE PROFILE ANALYSIS

- What is hot?
  - Can I make it faster, shrink the problem, parallelize it? (Not always...)
  - Reduce precision?
- What is cold?
  - Fill the gaps in the timeline
  - Can I take advantage of unused hardware?
  - Unnecessary dependencies or syncs?

# GENERAL OPTIMIZATION TIPS

- Using tensor cores?
  - Minimize conversions/transposes
- Increase grid and batch size to utilize GPU's 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

The screenshot displays the NVIDIA nsys performance analysis tool interface. The top section shows a 'Timeline View' with a horizontal axis representing time in milliseconds, ranging from +676,05ms to +676,3ms. Various API calls are visualized as colored bars: 'cudaStreams...' (green), 'elemen...' (blue), 'el...' (blue), 'cudaMemc...' (cyan), 'cudaMemcpyAsync' (red), and 'cuda...' (green). Below the timeline, 'OS runtime libraries' and 'CUDA HW' sections are visible, with 'pthread\_cond\_wait' highlighted in yellow. The bottom section is the 'Expert System View', which includes a table of events and a dropdown menu.

**Expert System View Table:**

Event Name	Duration	Start	Src Kind	Dst Kind	Bytes	PID	Device ID	Context ID	Stream ID	API Name
CUDA Async Memcpy with Pageable Memory	2,048 μs	6,38792s	Device	Pageable	8 B	75475		0	1	cudaMemcpy
	2,048 μs	6,8334s	Device	Pageable	4 B	75475		0	1	cudaMemcpy
	2,016 μs	2,5394s	Device	Pageable	4 B	75475		0	1	cudaMemcpy
	2,016 μs	3,90617s	Device	Pageable	48 B	75475		0	1	cudaMemcpy
	2,016 μs	4,25257s	Device	Pageable	4 B	75475		0	1	cudaMemcpy
	2,016 μs	5,67617s	Device	Pageable	48 B	75475		0	1	cudaMemcpy
	2,016 μs	5,9572s	Device	Pageable	8 B	75475		0	1	cudaMemcpy
	2,016 μs	5,97088s	Device	Pageable	4 B	75475		0	1	cudaMemcpy

**Expert System View Text:**

The following APIs use PAGEABLE memory which causes asynchronous CUDA memcpy operations to block and be executed synchronously. This leads to low GPU utilization.

Suggestion: If applicable, use PINNED memory instead.

CLI command:  
 nsys analyze -r cuda-async-memcpy /mnt/data/traces/qdrep/ncc1/profile\_circe-n011\_506451\_0.sqlite

**Dropdown Menu:**

- CUDA Async Memcpy with Pageable Memory
- CUDA Synchronous Memcpy
- CUDA Synchronous Memset
- CUDA Synchronization APIs
- CUDA GPU Starvation
- CUDA GPU Low Utilization
- VULKAN GPU Starvation
- VULKAN GPU Low Utilization



# NSIGHT COMPUTE

## KERNEL PROFILING TOOL

### 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, MacOSX (host only)

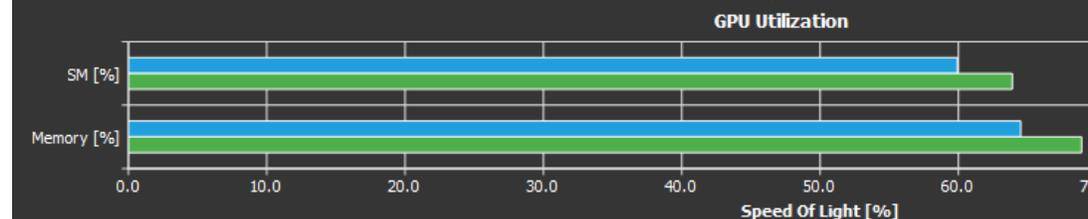
GPUs: Volta, Turing, Ampere GPUs

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 the theoretical maximum. High-level overview of the utilization for compute and memory resources of the GPU presented as a roofline chart.

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]



inst_executed [inst]	63,021,056 (284 instances)
l1tex__data_bank_conflicts_pipe_lsu_mem_shared_op_ld.sum	0
l1tex__data_bank_conflicts_pipe_lsu_mem_shared_op_st.sum	0
l1tex__data_bank_reads.avg.pct_of_peak_sustained_elapsed [%]	9.66
l1tex__data_bank_writes.avg.pct_of_peak_sustained_elapsed [%]	3.23
l1tex__data_pipe_lsu_wavefronts.avg.pct_of_peak_sustained_elapsed [%]	46.16
l1tex__data_pipe_lsu_wavefronts_mem_shared_cmd_read.sum	25,165,824
l1tex__data_pipe_lsu_wavefronts_mem_shared_cmd_read.sum.pct_of_peak_sustained_active [%]	40.75
l1tex__data_pipe_lsu_wavefronts_mem_shared_cmd_write.sum	2,097,152
l1tex__data_pipe_lsu_wavefronts_mem_shared_cmd_write.sum.pct_of_peak_sustained_active [%]	3.40
l1tex__data_pipe_tex_wavefronts.avg.pct_of_peak_sustained_elapsed [%]	0
l1tex__f_wavefronts.avg.pct_of_peak_sustained_elapsed [%]	0.00
l1tex__lsu_writeback_active.avg.pct_of_peak_sustained_elapsed [%]	42.59
l1tex__lsu_writeback_active.sum [cycle]	27,803,648
l1tex__lsu_writeback_active.sum.pct_of_peak_sustained_active [%]	45.03
l1tex__lsuin_requests.avg.pct_of_peak_sustained_elapsed [%]	66.00
l1tex__m_l1tex2xbar_req_cycles_active.avg.pct_of_peak_sustained_elapsed [%]	3.40
l1tex__m_l1tex2xbar_write_bytes.sum [Mbyte]	4.19
l1tex__m_l1tex2xbar_write_bytes_mem_global_op_red.sum [byte]	0



Targeted metric sections

Customizable data collection and presentation

Built-in expertise for Guided Analysis and optimization

The screenshot displays the NVIDIA Nsight Compute interface. At the top, the title bar reads "NVIDIA Nsight Compute". Below it is a menu bar with "File", "Connection", "Debug", "Profile", "Tools", and "Window". A toolbar contains icons for "Connect", "Disconnect", "Terminate", "Profile Kernel", and other controls. The main area shows a report for "old\_2\_fusion\_on\_softmax.nsisht-cuprof-report \*".

Key performance indicators are listed: "Current 64291 - softmax\_compute\_kernel (1966...)", "Time: 15.65 usecond", "Cycles: 16,235", "Regs: 28", "GPU: Tesla V100-SXM2-16GB", "SM Frequency: 1.04 cycle/nsecond", "CC: 7.0", and "Process: [944] python3.5".

The "GPU Speed Of Light" section provides a high-level overview and a table of metrics:

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

Below the table is a "GPU Utilization" bar chart showing "SM [%]" at approximately 46% and "Memory [%]" at approximately 43%. The x-axis is labeled "Speed Of Light [%]" and ranges from 0.0 to 100.0.

A "Bottleneck" warning is displayed: "[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."

The "Compute Workload Analysis" section provides a 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	Metric	Value
Executed Ipc Elapsed [inst/cycle]	1.83	SM Busy [%]	61.39
Executed Ipc Active [inst/cycle]	2.44	Issue Slots Busy [%]	61.39
Issued Ipc Active [inst/cycle]	2.46		-

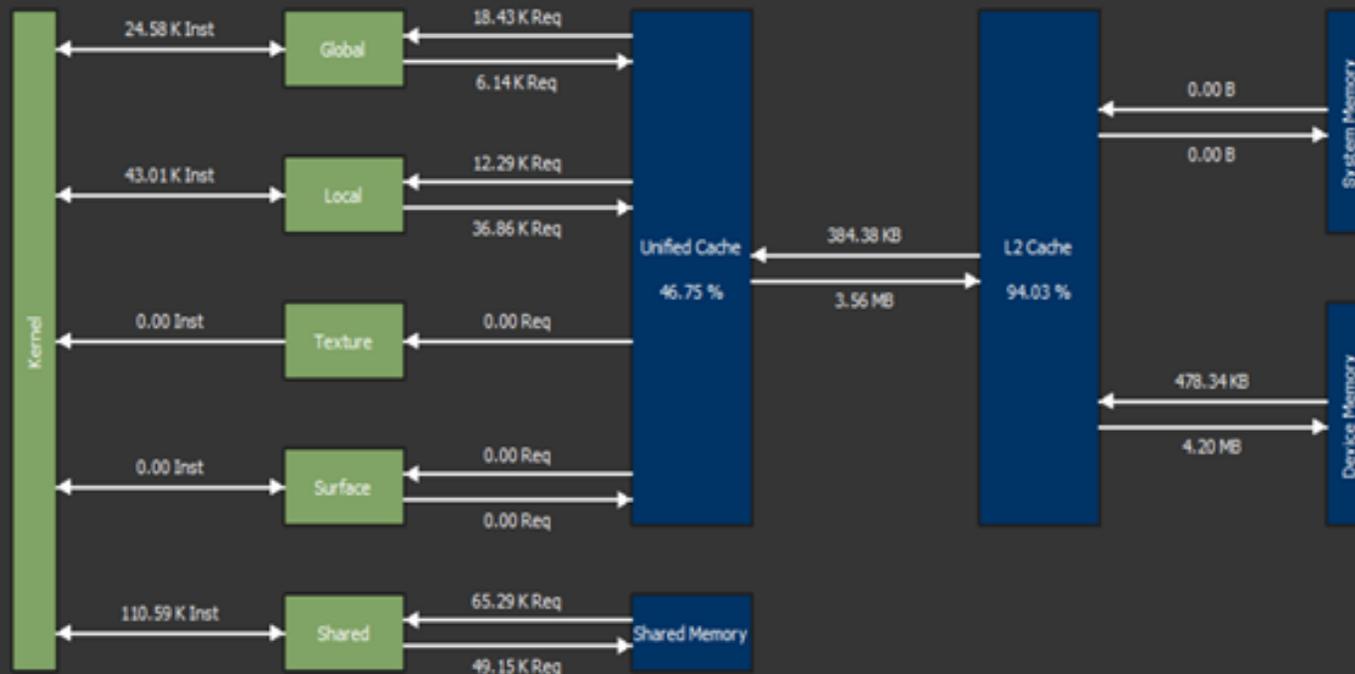
Memory Workload Analysis

All

Detailed analysis of the memory resources of the GPU. Memory can become a limiting factor for the overall kernel performance when fully utilizing the involved hardware units (Mem Busy), exhausting the available communication bandwidth between those units (Max Bandwidth), or by reaching the maximum throughput of issuing memory instructions (Mem Pipes Busy). Detailed chart of the memory units. Detailed tables with data for each memory unit.

Memory Throughput [Gbyte/second]	318.88	Mem Busy [%]	42.68
L1 Hit Rate [%]	46.75	Max Bandwidth [%]	44.73
L2 Hit Rate [%]	94.03	Mem Pipes Busy [%]	42.23

Memory Chart



Shared Memory

	Instructions	Requests	% Peak	Bank Conflicts
Shared Load	61,440	65,289	6.59	3,698
Shared Store	49,152	49,152	4.96	0
Shared Atomic	0	-	-	-
<b>Total</b>	<b>110,592</b>	<b>114,441</b>	<b>11.55</b>	<b>3,698</b>

First-Level (Unified) Cache

	Instructions	SM->TEX Requests	% Peak	Hit Rate	TEX->L2 Requests	% Peak	L2->TEX Returns	% Peak
Global Load Cached	18,432	18,432	1.86	66.65	-	-	-	-
Global Load Uncached	-	-	-	-	-	-	12,300	1.24
Local Load Cached	12,288	12,288	1.24	100	-	-	-	-

Visual memory analysis chart

Metrics for peak performance ratios

The screenshot displays the NVIDIA Nsight Compute interface, which is used for analyzing GPU performance. It is divided into several panels:

- Source Code Panel (Left):** Shows the C++ source code for a kernel named `softmax_compute_kernel`. The code includes variable declarations, loops, and CUDA-specific annotations like `__syncthreads()` and `cuda::Reduce1D`.
- Sampling Data Panel (Middle):** A table showing the execution of instructions. Each row includes an instruction ID, the source code line, and a 'Sampling Data (All)' column with a color-coded bar and a numerical value. For example, instruction 248 has a sampling data value of 111.
- Summary Panel (Center):** A yellow box provides a summary of the sampling data:
  - Total Sample Count: 111
  - Barrier: 43 (38.7%)
  - Mio Throttle: 21 (18.9%)
  - Not Selected: 8 (7.2%)
  - Selected: 7 (6.3%)
  - Short Scoreboard: 16 (14.4%)
  - Wait: 16 (14.4%)
- Sampling Data Panel (Right):** A second table showing a different set of instructions, such as `BSYNC B0`, `NOP`, and various `ISETP.GT.AND` instructions. Each row also includes a 'Sampling Data (All)' value and an 'Instructions Executed' value (e.g., 6,144).
- Metric Heatmap (Bottom Right):** A vertical heatmap on the right side of the interface, with a callout box stating: "Metric heatmap to quickly identify hotspots".

Source/PTX/SASS analysis and correlation

Source metrics per instruction

Metric heatmap to quickly identify hotspots

# KERNEL PROFILES WITH NSIGHT COMPUTE

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

- Generated file: report.ncu-rep
  - 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 displays the Standalone Source Viewer interface, which is used for viewing and correlating source code with assembly instructions for CUDA kernels. The interface is split into two main panes.

**Left Pane (Source Code):** Shows the source code for the kernel `device_tea_leaf_ppcg_solve_update_r`. The code is written in C++ and includes several constants and variables. The current line of code is highlighted in green:

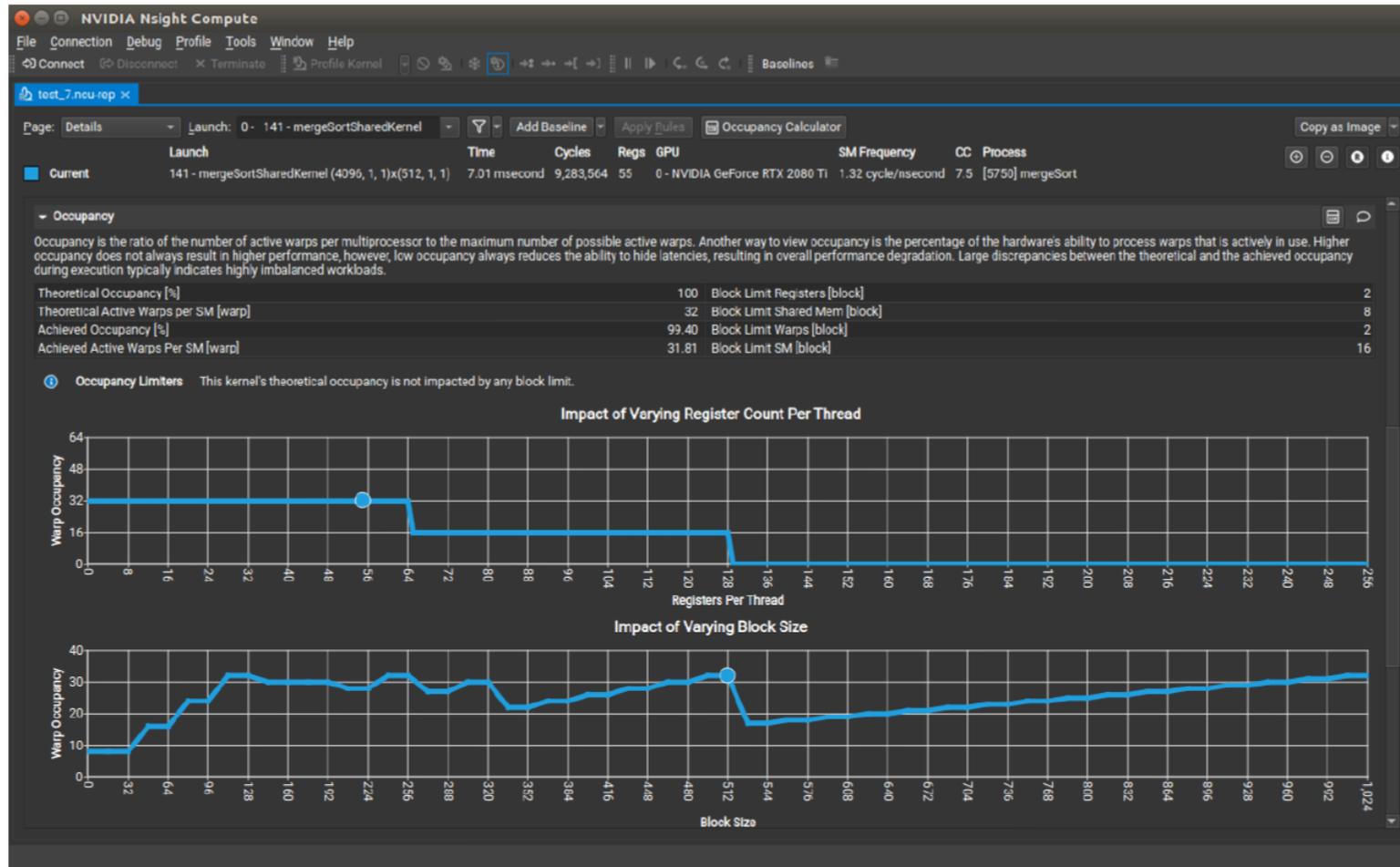
```
165 const double result = (1.0  
166 + (Ky[THARR2D(0, 1, 0)] + Ky[THARR2D(0, 0, 0)])  
167 + (Kx[THARR2D(1, 0, 0)] + Kx[THARR2D(0, 0, 0)])) + sd[THARR2D(0, 0, 0)]  
168 - (Ky[THARR2D(0, 1, 0)] * sd[THARR2D(0, 1, 0)] + Ky[THARR2D(0, 0, 0)] * sd[THARR2D(0, -1, 0)])  
169 - (Kx[THARR2D(1, 0, 0)] * sd[THARR2D(1, 0, 0)] + Kx[THARR2D(0, 0, 0)] * sd[THARR2D(-1, 0, 0)]);  
170
```

**Right Pane (Assembly):** Shows the assembly instructions corresponding to the source code. The instructions are listed with their addresses and sources:

```
22 00007f72 42fa6a59 IADD3 R5, R3, 0x1, RZ  
23 00007f72 42fa6a69 MOV R20, 0x8  
24 00007f72 42fa6a79 IMAD R21, R3, R2, R0  
25 00007f72 42fa6a89 IMAD R5, R2, R5, R0  
26 00007f72 42fa6a99 IMAD R3, R3, R2, -R2  
27 00007f72 42fa6aa9 IMAD.WIDE R16, R21, R20, c[0x0][0x1a0]  
28 00007f72 42fa6ab9 IADD3 R3, R0, R3, RZ  
29 00007f72 42fa6ac9 IMAD.WIDE R10, R5, R20, c[0x0][0x1a0]  
30 00007f72 42fa6ad9 IMAD.WIDE R18, R3, R20, c[0x0][0x1a0]  
31 00007f72 42fa6ae9 LDG.E.64.CONSTANT.SYS R16, [R16]  
32 00007f72 42fa6af9 IMAD.WIDE R26, R21, R20, c[0x0][0x190]  
33 00007f72 42fa6b09 LDG.E.64.CONSTANT.SYS R10, [R10]  
34 00007f72 42fa6b19 IMAD.WIDE R28, R21, R20, c[0x0][0x1a0]  
35 00007f72 42fa6b29 LDG.E.64.CONSTANT.SYS R18, [R18]  
36 00007f72 42fa6b39 IMAD.WIDE R12, R5, R20, c[0x0][0x1a0]  
37 00007f72 42fa6b49 LDG.E.64.CONSTANT.SYS R14, [R26]  
38 00007f72 42fa6b59 LDG.E.64.CONSTANT.SYS R6, [R26+0x8]  
39 00007f72 42fa6b69 LDG.E.64.CONSTANT.SYS R2, [R28+0x0]  
40 00007f72 42fa6b79 LDG.E.64.CONSTANT.SYS R12, [R12]  
41 00007f72 42fa6b89 LDG.E.64.CONSTANT.SYS R8, [R28+0x8]  
42 00007f72 42fa6b99 LDG.E.64.CONSTANT.SYS R4, [R28]  
43 00007f72 42fa6ba9 IMAD.WIDE R20, R21, R20, c[0x0][0x190]  
44 00007f72 42fa6bb9 LDG.E.64.SYS R22, [R20]  
45 00007f72 42fa6bc9 DADD R24, R16, R10  
46 00007f72 42fa6bd9 DMIII R18, R16, R18
```

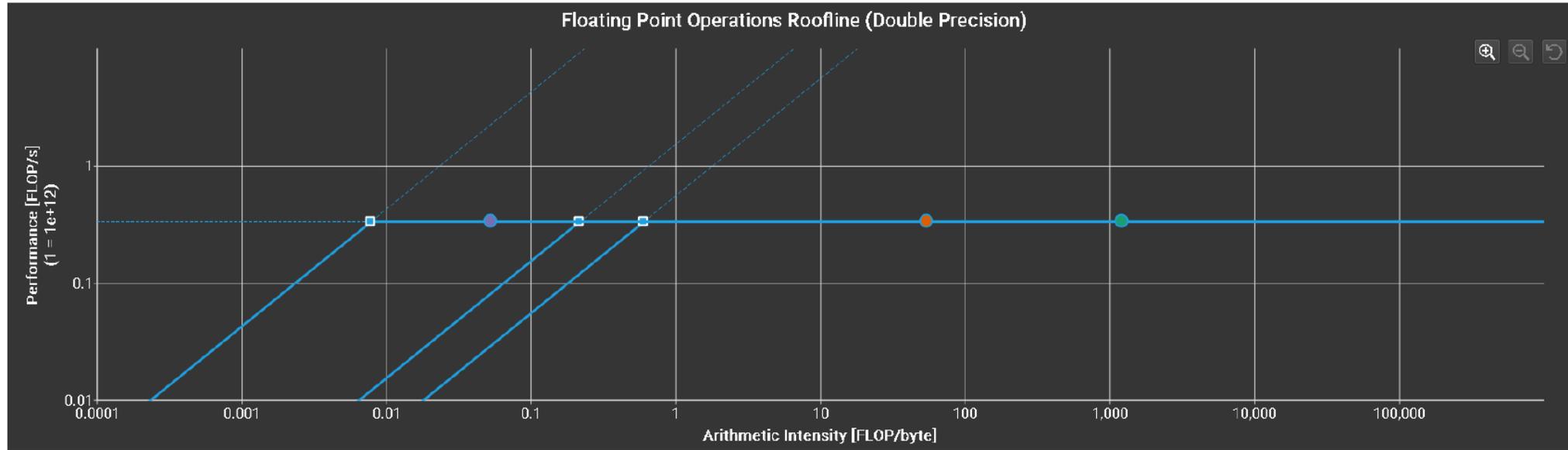
# OCCUPANCY CALCULATOR

## 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

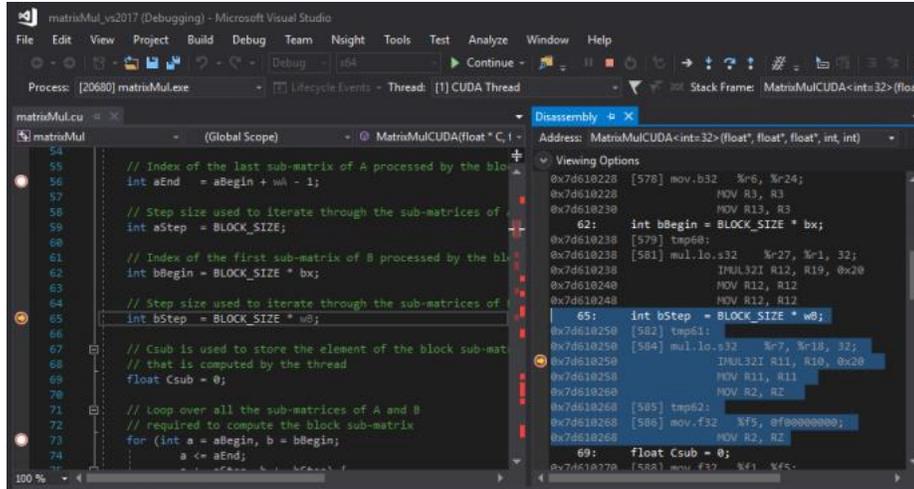


- Visualize multiple levels of the memory hierarchy
- Identify bottlenecks caused by memory limitations
- Determine how modifying algorithms may (or may not) impact performance

Sections/Rules Info			
Sections/Rules			
Enter filter			
Name	Priority	Description	
<input checked="" type="checkbox"/> GPU Speed Of Light Throughput (1)	10	High-level overview of the throughput for compu...	
<input checked="" type="checkbox"/> GPU Speed Of Light Roofline Chart (1)	11	High-level overview of the utilization for comput...	
<input checked="" type="checkbox"/> GPU Speed Of Light Hierarchical Roofline Chart (Double Precision)	12	High-level overview of the utilization for comp...	
<input checked="" type="checkbox"/> GPU Speed Of Light Hierarchical Roofline Chart (Half Precision)	12	High-level overview of the utilization for comput...	
<input checked="" type="checkbox"/> GPU Speed Of Light Hierarchical Roofline Chart (Single Precision)	12	High-level overview of the utilization for comput...	
<input checked="" type="checkbox"/> GPU Speed Of Light Hierarchical Roofline Chart (Tensor Core)	12	High-level overview of the utilization for comput...	
<input type="checkbox"/> Compute Workload Analysis (2)	20	Detailed analysis of the compute resources of t...	

# DEVELOPER TOOLS

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



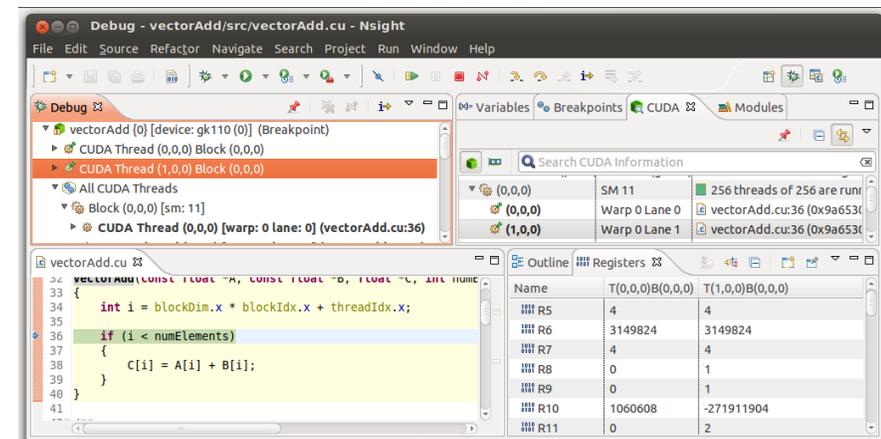
**Profilers:** Nsight Systems, Nsight Compute, CUPTI, NVIDIA Tools eXtension (NVTX)



**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_kernl i(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



# CUDA GDB

## COMMAND LINE AND IDE BACKEND DEBUGGER

- Unified CPU and CUDA Debugging
- CUDA-C/PTX/SASS support
- Built on GDB and uses many of the same CLI commands

```
(cuda-gdb) info cuda threads breakpoint all
  BlockIdx ThreadIdx           Virtual PC Dev SM Wp Ln           Filename Line
Kernel 0
  (1,0,0)   (0,0,0) 0x0000000000948e58    0 11  0  0 infoCommands.cu  12
  (1,0,0)   (1,0,0) 0x0000000000948e58    0 11  0  1 infoCommands.cu  12
  (1,0,0)   (2,0,0) 0x0000000000948e58    0 11  0  2 infoCommands.cu  12
  (1,0,0)   (3,0,0) 0x0000000000948e58    0 11  0  3 infoCommands.cu  12
  (1,0,0)   (4,0,0) 0x0000000000948e58    0 11  0  4 infoCommands.cu  12
  (1,0,0)   (5,0,0) 0x0000000000948e58    0 11  0  5 infoCommands.cu  12

(cuda-gdb) info cuda threads breakpoint 2 lane 1
  BlockIdx ThreadIdx           Virtual PC Dev SM Wp Ln           Filename Line
Kernel 0
  (1,0,0)   (1,0,0) 0x0000000000948e58    0 11  0  1 infoCommands.cu  12
```

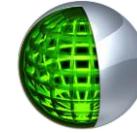
# COMPUTE SANITIZER

## AUTOMATICALLY SCAN FOR BUGS AND MEMORY ISSUES

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

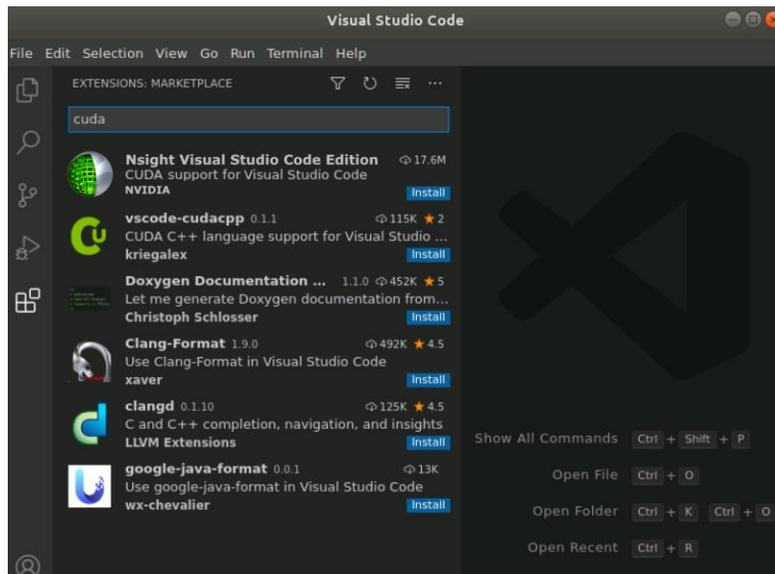
```
$ compute-sanitizer --leak-check full memcheck_demo
===== COMPUTE-SANITIZER
Mallocing memory
Running unaligned_kernel
Ran unaligned_kernel: no error
===== Invalid __global__ write of size 4 bytes
=====   at 0x160 in memcheck_demo.cu:6:unaligned_kernel()
=====   by thread (0,0,0) in block (0,0,0)
=====   Address 0x7fa73cc00001 is misaligned
=====   Saved host backtrace up to driver entry point at kernel launch time
=====   Host Frame: [0x2068ea]
=====             in /usr/lib/x86_64-linux-gnu/libcuda.so.1
=====   Host Frame: [0xfc4b]
=====             in memcheck_demo
=====   Host Frame: [0x66288]
=====             in memcheck_demo
=====   Host Frame: [0xb4eb]
=====             in memcheck_demo
=====   Host Frame: __device_stub__Z16unaligned_kernelv(void) [0xb1fc]
=====             in memcheck_demo
=====   Host Frame: unaligned_kernel(void) [0xb257]
=====             in memcheck_demo
=====   Host Frame: run_unaligned(void) [0xaf38]
=====             in memcheck_demo
=====   Host Frame: main [0xb0b1]
=====             in memcheck_demo
=====   Host Frame: __libc_start_main [0x28cb2]
=====             in /lib/x86_64-linux-gnu/libc.so.6
=====   Host Frame: start [0xad9e]
=====             in memcheck_demo
```

# 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!



Variables view

CPU & GPU registers

Watch CPU & GPU vars

Session status

Exec debugger commands

CUDA Call Stack

CUDA focus

```
matrixMul.cu
06 // Csub is used to store the element of the block sub-matrix
07 // that is computed by the thread
08 float Csub = 0;
09
10 // Loop over all the sub-matrices of A and B
11 // required to compute the block sub-matrix
12 for (int a = aBegin, b = bBegin; a <= aEnd; a += aStep, b += bStep) {
13     // Declaration of the shared memory array As used to
14     // store the sub-matrix of A
15     __shared__ float As[BLOCK_SIZE][BLOCK_SIZE];
16
17     // Declaration of the shared memory array Bs used to
18     // store the sub-matrix of B
19     __shared__ float Bs[BLOCK_SIZE][BLOCK_SIZE];
20
21     // Load the matrices from device memory
22     // to shared memory; each thread loads
23     // one element of each matrix
24     As[ty][tx] = A[a + wA * ty + tx];
25     Bs[ty][tx] = B[b + wB * ty + tx];
26
27     // Synchronize to make sure the matrices are loaded
28     syncthreads();
29
30     // Multiply the two matrices together;
31     // each thread computes one element
32     // of the block sub-matrix
33     #pragma unroll
34     for (int k = 0; k < BLOCK_SIZE; k++) {
35         Csub += As[ty][k] * Bs[k][tx];
36     }
37 }
```

Device	SM	Wp	Active	Lanes	Mask	Active	Physical	PC	Kernel	BlockIdx	First	Active	ThreadIdx
0	1	0x00000000	0x00000000	0x00000000	0x00000000	0	(1,0,0)	(0,0,0)					
1	1	0xffffffff	0x00000000	0x00000000	0x00000000	0	(1,0,0)	(0,1,0)					
2	1	0xffffffff	0x00000000	0x00000000	0x00000000	0	(1,0,0)	(0,2,0)					
3	1	0xffffffff	0x00000000	0x00000000	0x00000000	0	(1,0,0)	(0,3,0)					
4	1	0xffffffff	0x00000000	0x00000000	0x00000000	0	(1,0,0)	(0,4,0)					
5	1	0xffffffff	0x00000000	0x00000000	0x00000000	0	(1,0,0)	(0,5,0)					
6	1	0xffffffff	0x00000000	0x00000000	0x00000000	0	(1,0,0)	(0,6,0)					
7	1	0xffffffff	0x00000000	0x00000000	0x00000000	0	(1,0,0)	(0,7,0)					

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

### NASA GPU Hackathon 2022

Date(s): Sep 19, 2022 - Sep 28, 2022

Event Focus: **HPC+AI**

North America/Latin America



Applications **Closed**

### CSCS GPU Hackathon 2022

Date(s): Sep 19, 2022 - Sep 29, 2022

Event Focus: **HPC+AI**

Europe/Middle East/Africa



Applications **Closed**

### NVIDIA/HLRS SciML GPU Bootcamp

Date(s): Oct 24, 2022 - Oct 25, 2022

Event Focus: **HPC**

Europe/Middle East/Africa



HLRS



Applications **Open**

### OLCF GPU Hackathon 2022

Date(s): Oct 17, 2022 - Oct 27, 2022

Event Focus: **HPC+AI**

North America/Latin America



Applications **Open**

### NERSC GPU Hackathon 2022

Date(s): Nov 30, 2022 - Dec 8, 2022

Event Focus: **HPC+AI**

North America/Latin America



Applications **Open**

# 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