DEVITO WITH CUDA

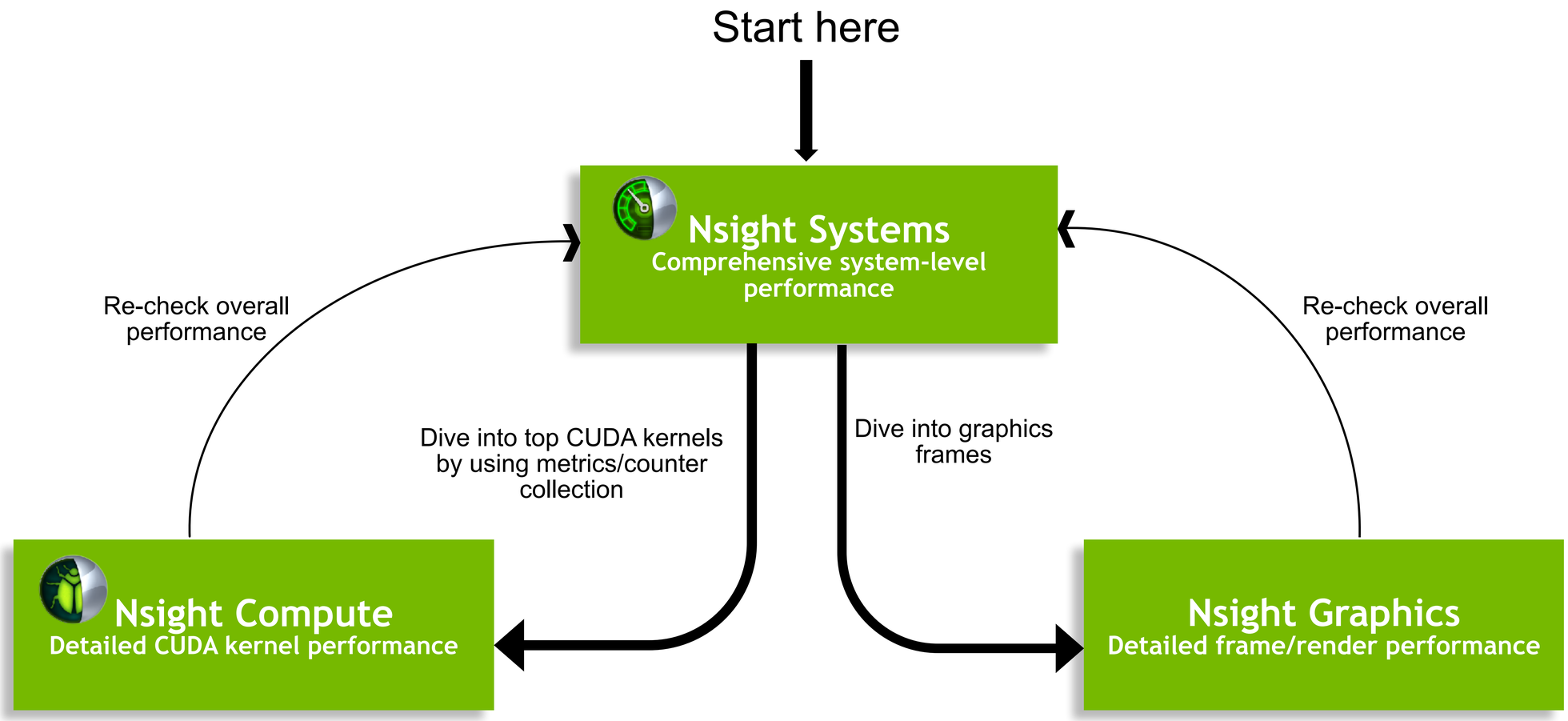
FLORENT DUGUET

AGENDA

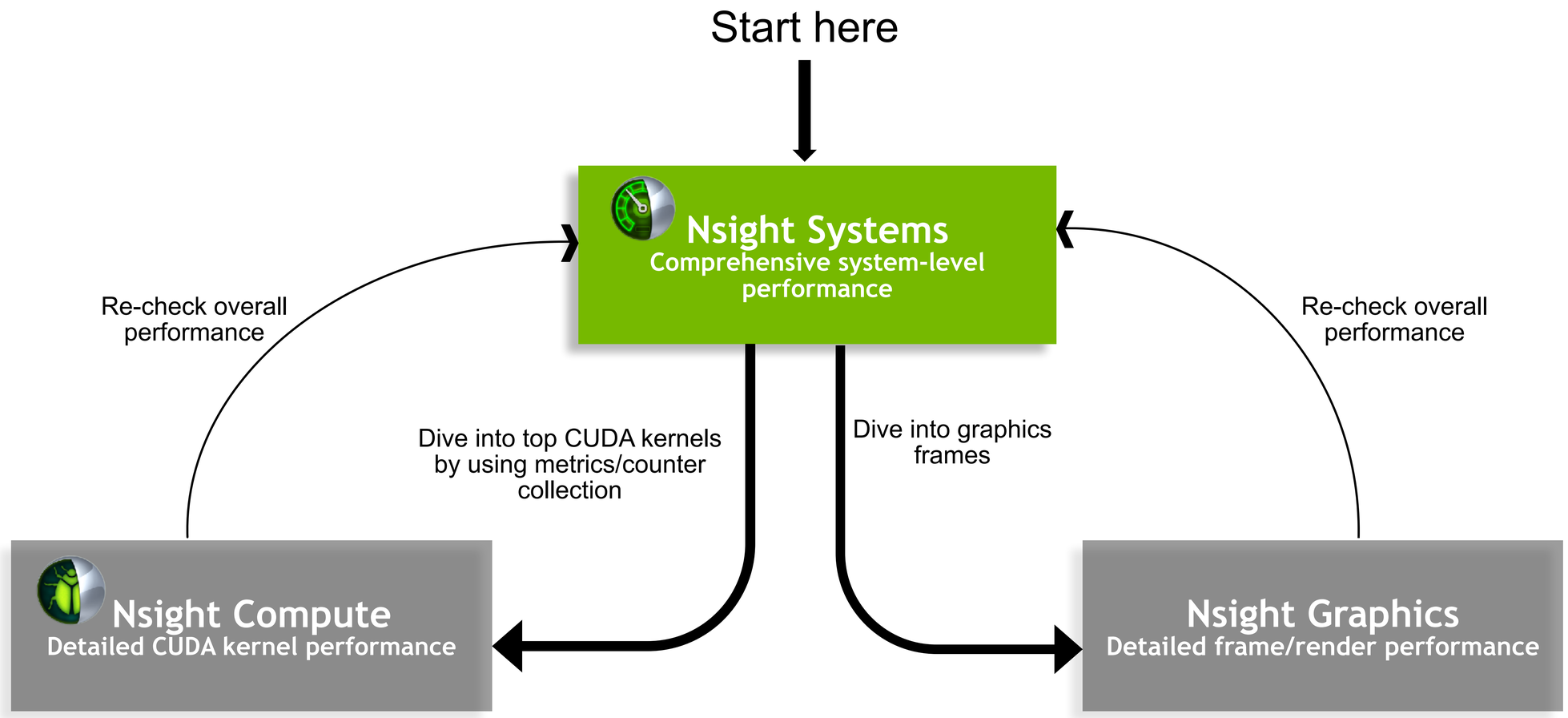
➢ Using Nsight Compute profiler

- ❖ Intro to Profile Devito generated Code with Nsight Compute
- ❖ Exercise 1: Coalesced Memory
- ❖ Exercise 2: Block Size
- ❖ Exercise 3: Reduce Memory Reads
- ❖ Exercise 4 : Maximize cache hits
- ❖ Exercise 5: Prefetch with Shared Memory

NSIGHT TOOLS WORKFLOW



NSIGHT TOOLS WORKFLOW



NSIGHT SYSTEMS

System profiler

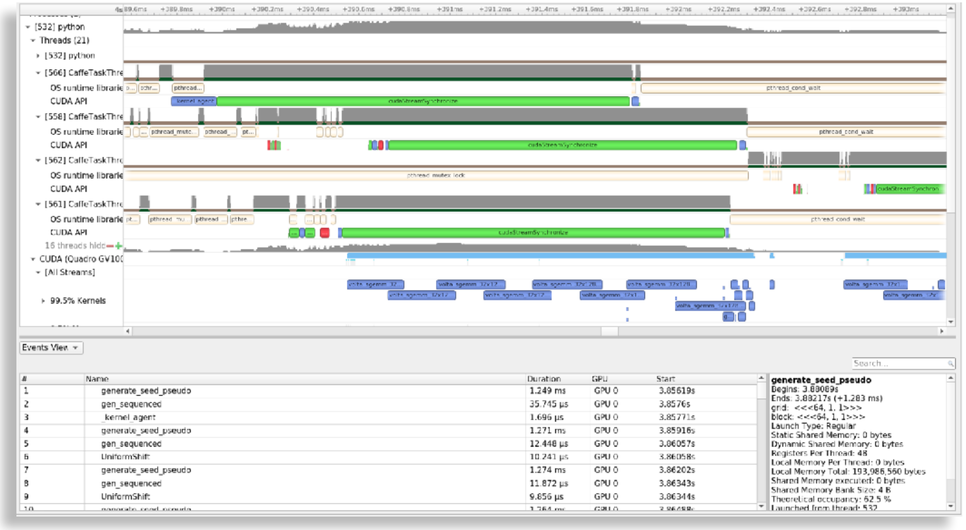
Key Features:

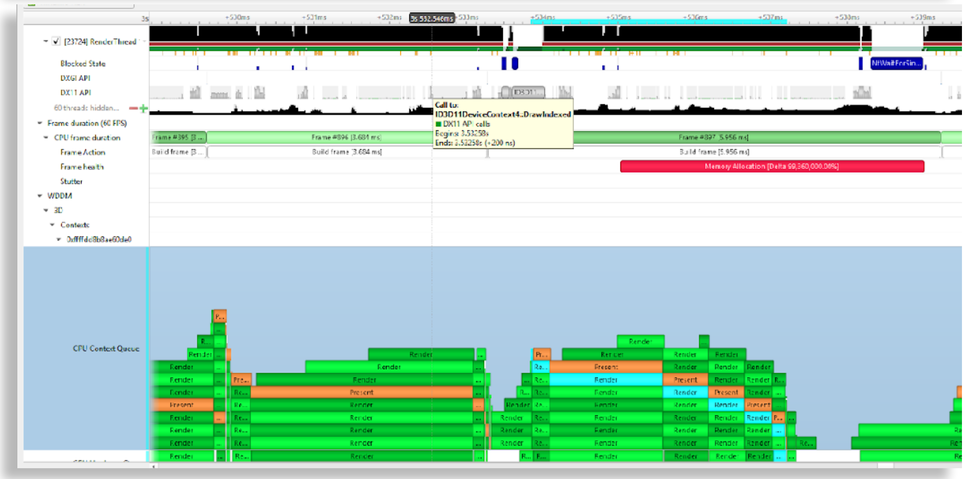
- System-wide application algorithm tuning
  - Multi-process tree support
- Locate optimization opportunities
  - Visualize millions of events on a very fast GUI timeline
  - Or gaps of unused CPU and GPU time
- Balance your workload across multiple CPUs and GPUs
  - CPU algorithms, utilization and thread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

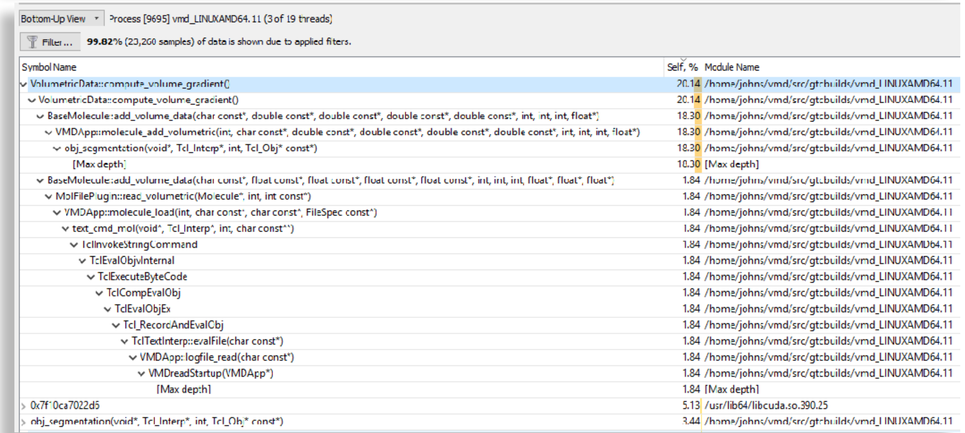
OS: Linux (x86, **Power**, Arm SBSA, Tegra), Windows, MacOSX (host)

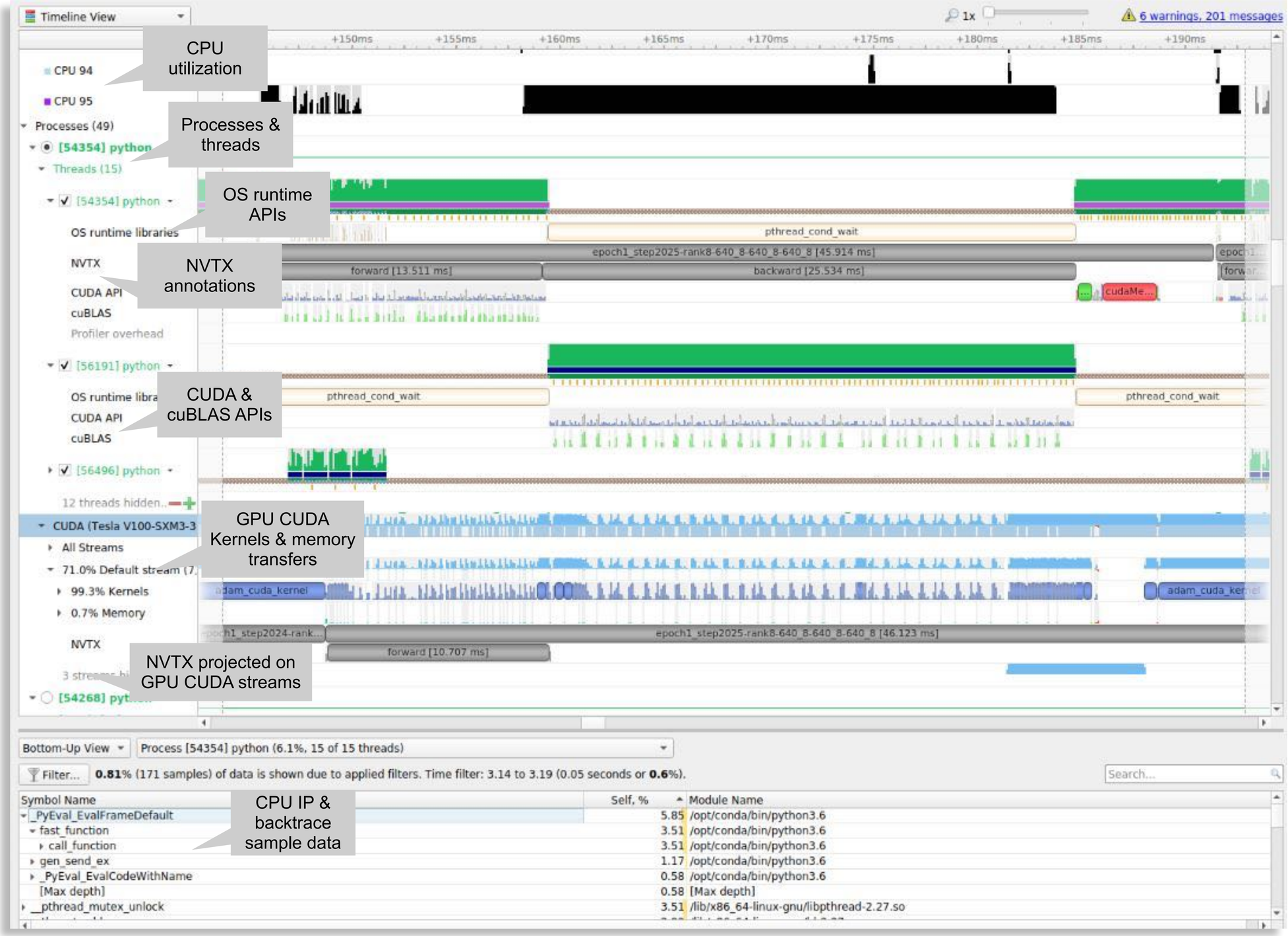
GPUs: Pascal+

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









Additionally

Trace:
- TensorRT
- Direct3D11,12,DXR
- Vulkan
- OpenGL
- OpenACC
- MPI
- OpenMP
- Ftrace
- ETW
- WDDM
- GPU Context Switch

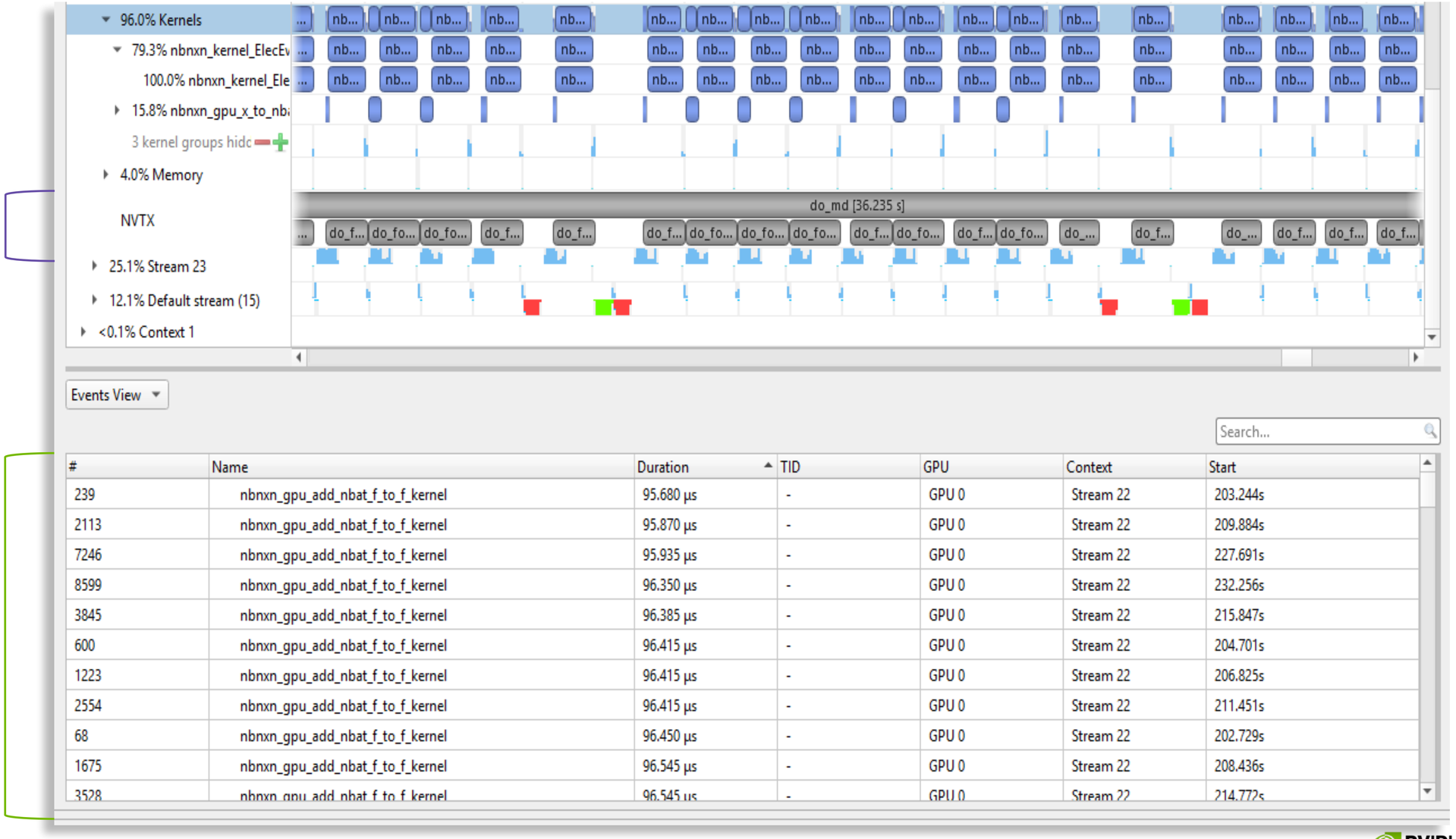
Export:
- SQLite
- HDF5
- JSON

Architectures:
- X86_64
- Power
- Arm SBSA
- Tegra

NSIGHT SYSTEMS

Key Features

- NVTX projected onto GPU
- Event table



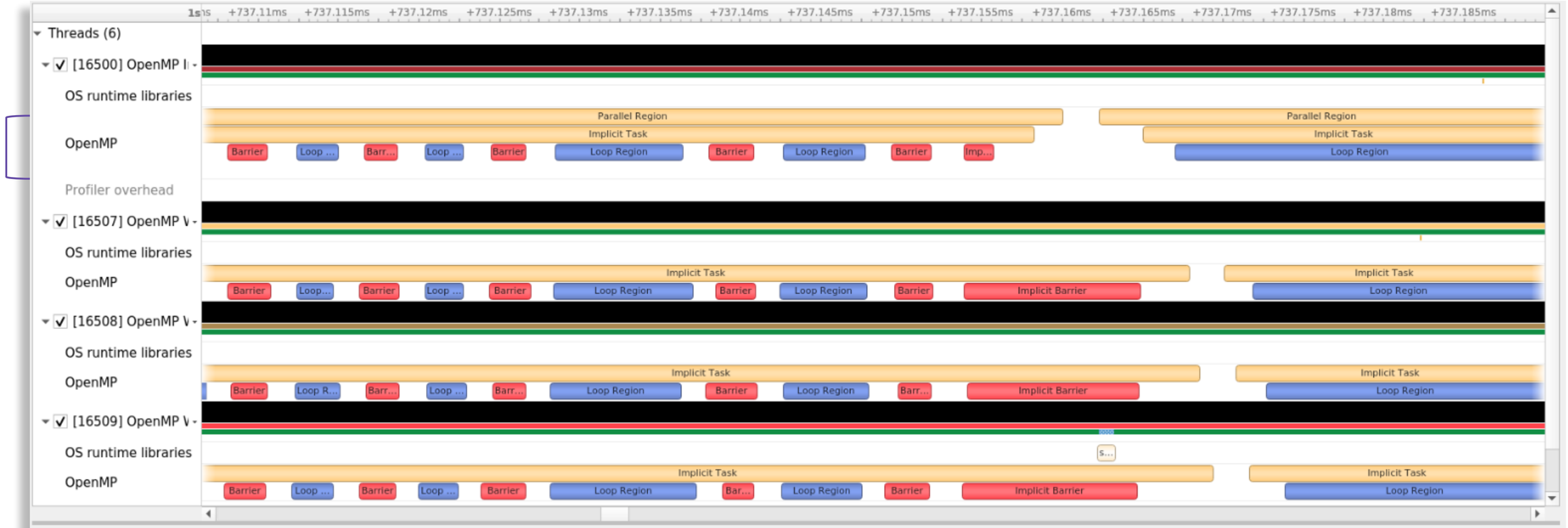
#	Name	Duration	TID	GPU	Context	Start
239	nbnxn_gpu_add_nbat_f_to_f_kernel	95.680 µs	-	GPU 0	Stream 22	203.244s
2113	nbnxn_gpu_add_nbat_f_to_f_kernel	95.870 µs	-	GPU 0	Stream 22	209.884s
7246	nbnxn_gpu_add_nbat_f_to_f_kernel	95.935 µs	-	GPU 0	Stream 22	227.691s
8599	nbnxn_gpu_add_nbat_f_to_f_kernel	96.350 µs	-	GPU 0	Stream 22	232.256s
3845	nbnxn_gpu_add_nbat_f_to_f_kernel	96.385 µs	-	GPU 0	Stream 22	215.847s
600	nbnxn_gpu_add_nbat_f_to_f_kernel	96.415 µs	-	GPU 0	Stream 22	204.701s
1223	nbnxn_gpu_add_nbat_f_to_f_kernel	96.415 µs	-	GPU 0	Stream 22	206.825s
2554	nbnxn_gpu_add_nbat_f_to_f_kernel	96.415 µs	-	GPU 0	Stream 22	211.451s
68	nbnxn_gpu_add_nbat_f_to_f_kernel	96.450 µs	-	GPU 0	Stream 22	202.729s
1675	nbnxn_gpu_add_nbat_f_to_f_kernel	96.545 µs	-	GPU 0	Stream 22	208.436s
3528	nbnxn_gpu_add_nbat_f_to_f_kernel	96.545 µs	-	GPU 0	Stream 22	214.772s

NSIGHT SYSTEMS

MPI & OpenACC Trace

NSIGHT SYSTEMS

OpenMP 5 API Trace



Events View

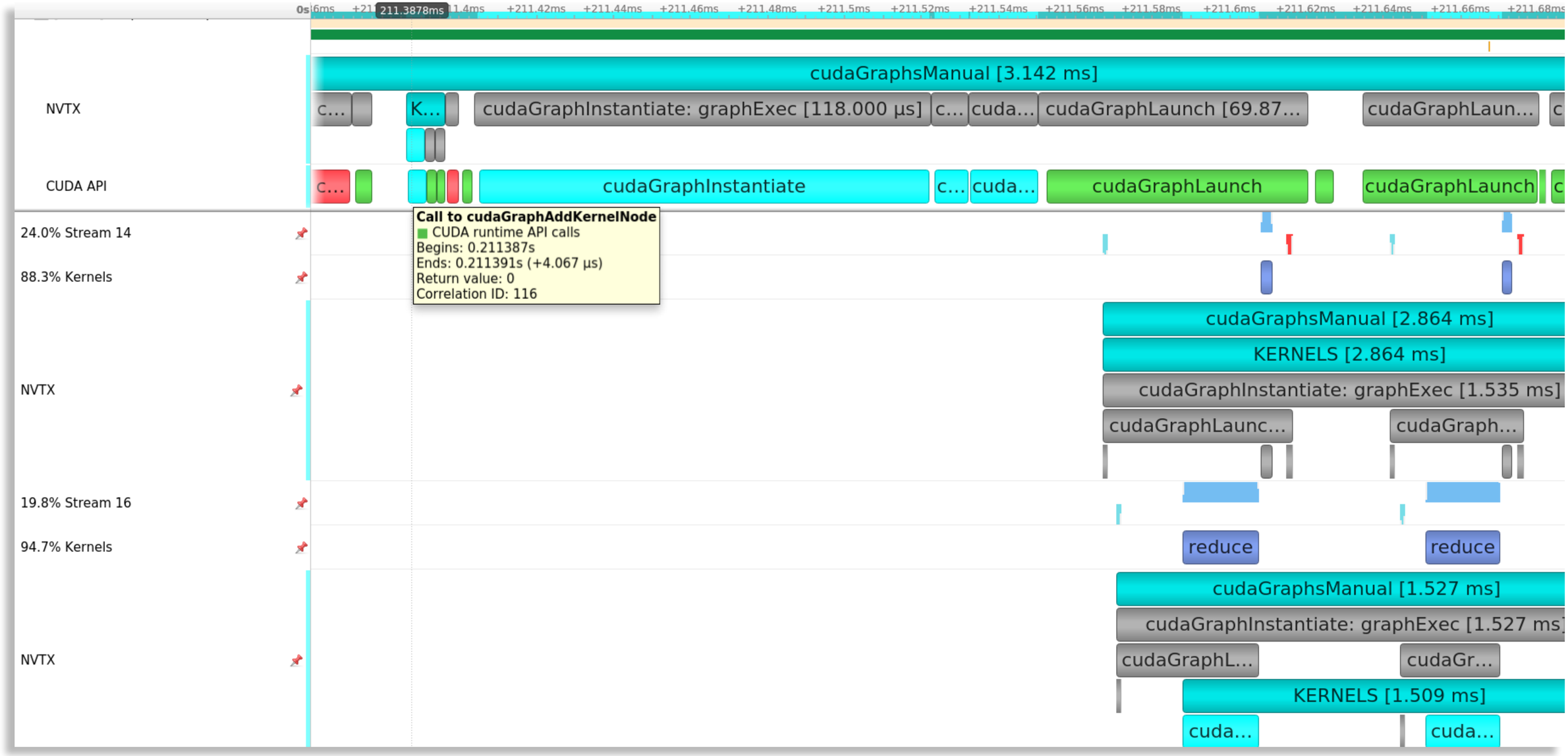
Search...

#	Name	Duration	TID	Start
1	Parallel Region	3.943 ms	16500	0.0698709s
2	Implicit Task	2.831 ms	16500	0.0709764s
3	Waiting to enter Critical Region	3.019 µs	16500	0.0709882s
4	Critical Region	5.410 µs	16500	0.0709912s
5	Waiting for Nested Lock	2.852 µs	16500	0.0710003s
6	Nested Lock Released	-	16500	0.0713706s
7	Waiting for Nested Lock	1.907 µs	16500	0.0713733s
8	Nested Lock Released	-	16500	0.0714109s
9	Waiting for Nested Lock	1.897 µs	16500	0.071413s
10	Nested Lock Released	-	16500	0.0714193s
11	Waiting for Nested Lock	1.921 µs	16500	0.0714212s

Waiting to enter Critical Region
Begins: 0.0709882s
Ends: 0.0709912s (+3.019 µs)
Task Id: 2
Kind: Critical Region

NSIGHT SYSTEMS

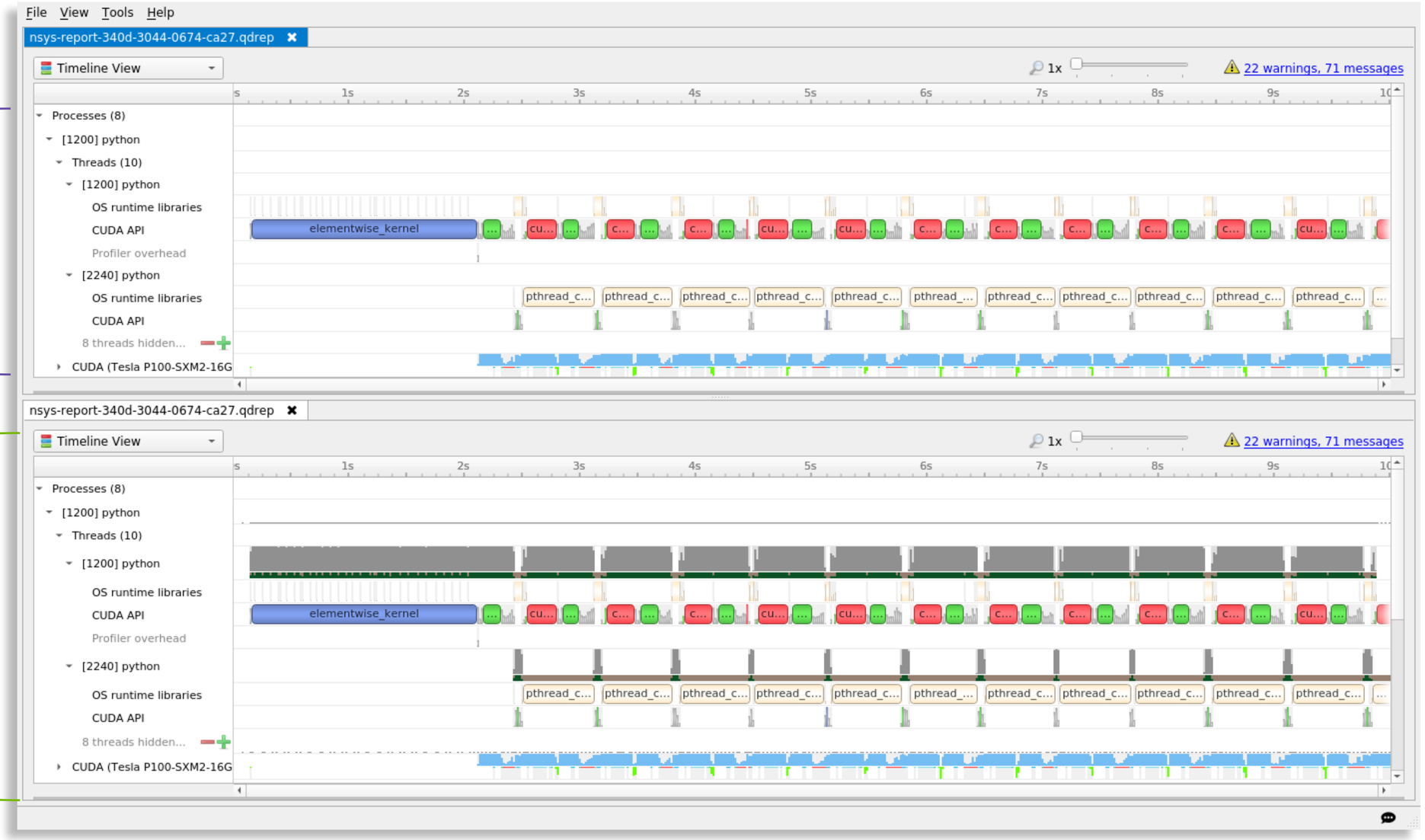
CUDA graph node correlation & NVTX projection



NSIGHT SYSTEMS

Improved UX without thread scheduler data permission

- Before
  - No utilization
  - No sorting
- After
  - Clear activity/state
  - Sort by utilization



NSIGHT SYSTEMS

Other Features

CLI improvement

- Simultaneous sessions & management
- `nsys launch` forwards terminal and signals
- `nsys stats` command
- `nsys export` command

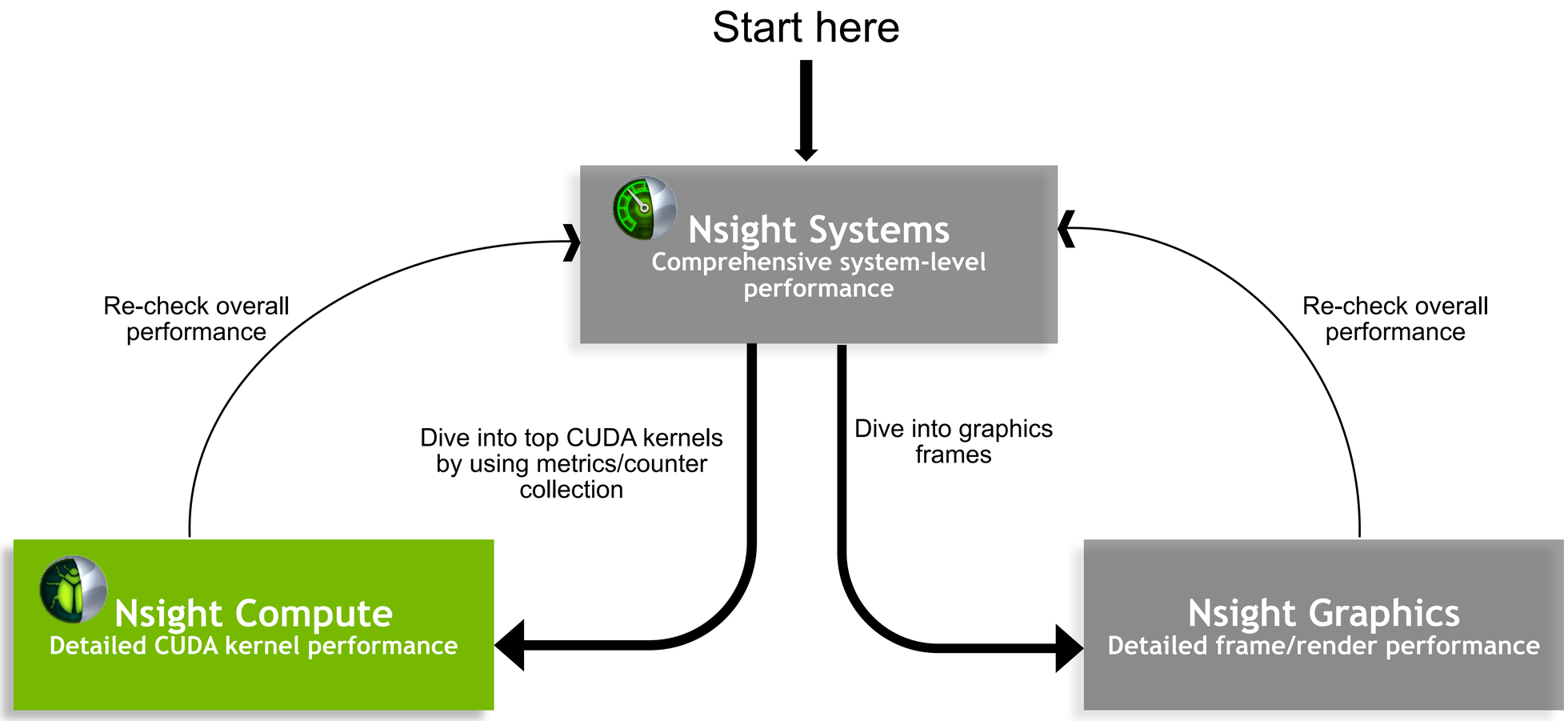
CUDA context “All Streams” aggregation tree

Windows WDDM GPU memory usage, paging, evictions

Graphics trace enhancements

```
1 2 3 4 5
Terminal
16:23:58 $
16:24:07 $ nsys profile --output /tmp/smoke --show-output false ./smokeParticles
Collecting data...

Terminal
16:24:18 $ nsys start --session-new particles --output /tmp/particles
start executed. use the 'launch' command to start the collection
16:24:22 $ nsys launch --session particles --show-output false ./particles
application launched

The target application terminated with signal 15 (SIGTERM)

Terminal
16:23:51 $
16:24:28 $ nsys launch --trace cuda --sample none --session-new fog --show-output false ./randomFog
WARNING: Backtraces will not be collected because sampling is disabled.
application launched

Terminal
              ID         TIME                STATE LAUNCH NAME
         1016352        00:25           Collecting      1 profile-smokeParticles-16352
         1116423        00:15           Collecting      1 particles
         1216488        00:04          WaitForStart      1 fog
16:24:40 $ nsys cancel --session 1016352
cancel executed
16:24:42 $ nsys sessions list
              ID         TIME                STATE LAUNCH NAME
         1016352        00:31          WaitForStart      1 profile-smokeParticles-16352
         1116423        00:21            Collecting      1 particles
         1216488        00:10          WaitForStart      1 fog
16:24:46 $ nsys start --output /tmp/fog --session fog
start executed
16:24:50 $ nsys stop --session fog
Processing events...
Saving intermediate "/tmp/fog.qdstrm" file to disk...

Importing [==============================================================100%]
Saved report file to "/tmp/fog.qdrep"
stop executed
16:24:54 $ nsys sessions list
              ID         TIME                STATE LAUNCH NAME
         1016352        00:43          WaitForStart      1 profile-smokeParticles-16352
         1116423        00:33            Collecting      1 particles
         1216488        00:22          WaitForStart      1 fog
16:24:57 $ nsys shutdown --session particles
collection cancelled. shutdown executed
16:25:01 $ nsys sessions list
              ID         TIME                STATE LAUNCH NAME
         1016352        00:50          WaitForStart      1 profile-smokeParticles-16352
         1216488        00:29          WaitForStart      1 fog
```

NSIGHT TOOLS WORKFLOW



NSIGHT COMPUTE

Kernel Profiling Tool

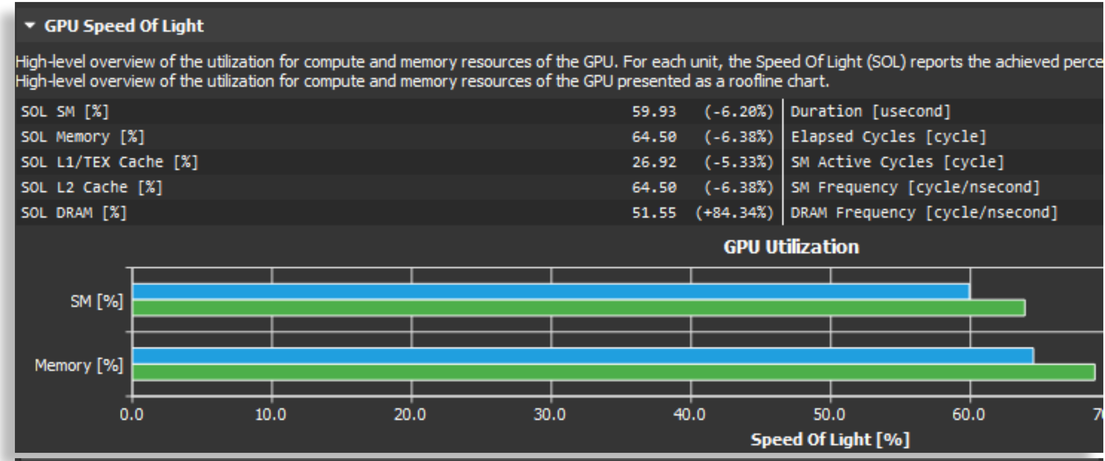
Key Features:

- Interactive CUDA API debugging and kernel profiling
- Fast Data Collection
- Compare performance metrics across different runs
- Fully Customizable (Programmable UI/Guided Analysis)
- Command Line, Standalone, IDE Integration

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

GPUs: Volta, Turing, A100 GPUs

Docs/product: https://developer.nvidia.com/nsight-compute



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

@P0 EXIT	6	108	49	1,404,672
IADD3 R7, P2, R0, UR7, RZ	7	177	95	1,401,344
IADD3 R6, P1, R4, UR4, RZ	8	30	9	1,401,344
ISETP.GE.U32.AND P0, PT, R7, UR5, PT	8	12	5	1,401,344
IADD3.X R8, R2, UR8, RZ, P2, !PT	8	14	3	1,401,344
IMAD.X R7, RZ, RZ, R5, P1	9	9	0	1,401,344
ISETP.GE.U32.AND.EX P0, PT, R8, UR6, PT, P0	9	106	35	1,401,344
STG.E.U8 [R6.64], R3	8	116	75	1,401,344
@P0 EXIT	8	92	33	1,401,344
IADD3 R8, P2, R0, UR9, RZ	9	45	14	1,397,120
IADD3 R6, P1, R6, UR4, RZ	9	248	145	1,397,120
ISETP.GE.U32.AND P0, PT, R8, UR5, PT	9	57	17	1,397,120
IADD3.X R8, R2, UR12, RZ, P2, !PT	9	11	4	1,397,120
IMAD.X R7, RZ, RZ, R7, P1	9	7	0	1,397,120
ISETP.GE.U32.AND.EX P0, PT, R8, UR6, PT, P0	9	94	15	1,397,120
STG.E.U8 [R6.64], R3	8	104	61	1,397,120

NSIGHT COMPUTE

Profile Report – Details Page

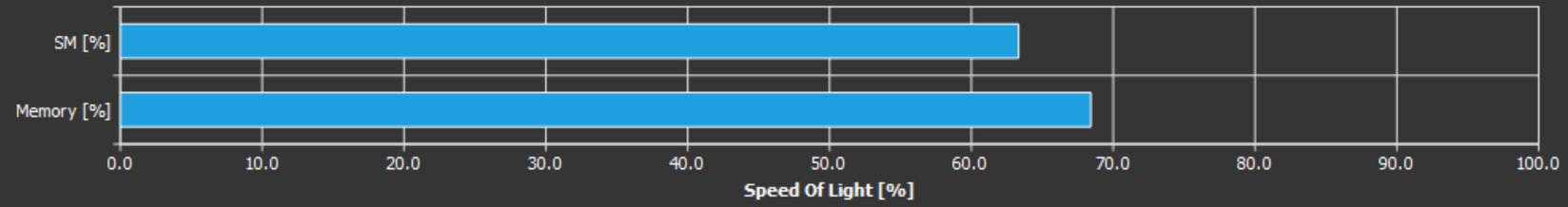
All Data on Single Page

GPU Speed Of Light — SOL Chart

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

SOL SM [%]	63.33	Duration [usecond]	476.51
SOL Memory [%]	68.43	Elapsed Cycles [cycle]	561,700
SOL L1/TEX Cache [%]	28.27	SM Active Cycles [cycle]	503,207.61
SOL L2 Cache [%]	68.43	SM Frequency [cycle/nsecond]	1.18
SOL DRAM [%]	27.73	DRAM Frequency [cycle/nsecond]	1.59

GPU Utilization



Recommendations

ell-balanced: To reduce runtime, both computation and memory traffic must be reduced. Check both the Compute Workload Analysis and Memory tions.

Roofline Analysis The ratio of peak float (fp32) to double (fp64) performance on this device is 2:1. The kernel achieved 0% of this device's fp32 peak performance and 0% of its fp64 peak performance.

Focused Sections

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.

Executed Ipc Elapsed [inst/cycle]	2.03	SM Busy [%]	70.66
Executed Ipc Active [inst/cycle]	2.27	Issue Slots Busy [%]	56.68
Issued Ipc Active [inst/cycle]	2.27	-	-

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 betwee Mem Pipes Busy). Detailed chart of the memory units. Detailed tables with data for each memory

Memory Throughput [Gbyte/second]	68.43
L1/TEX Hit Rate [%]	33.92
L2 Hit Rate [%]	25.34
L2 Compression Success Rate [%]	4

Ordered from Top-Level to Low-Level

NVIDIA

NSIGHT COMPUTE

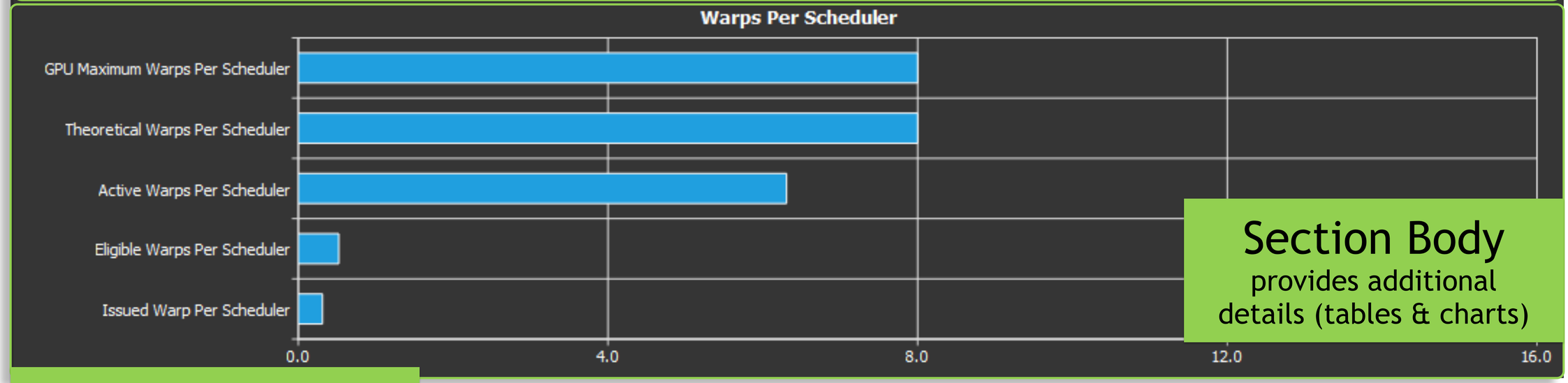
Section Example

Section Header
provides overview & context for other sections

Scheduler Statistics

Summary of the activity of the schedulers issuing instructions. Each scheduler maintains a pool of warps that it can issue instructions for. The upper bound of warps in the pool (Theoretical Warps) is limited by the launch configuration. On every cycle each scheduler checks the state of the allocated warps in the pool (Active Warps). Active warps that are not stalled (Eligible Warps) are ready to issue their next instruction. From the set of eligible warps the scheduler selects a single warp from which to issue one or more instructions (Issued Warp). On cycles with no eligible warps, the issue slot is skipped and no instruction is issued. Having many skipped issue slots indicates poor latency hiding.

Active Warps Per Scheduler [warp]	6.31	Instructions Per Active Issue Slot [inst/cycle]	1
Eligible Warps Per Scheduler [warp]	0.52	No Eligible [%]	68.74
Issued Warp Per Scheduler	0.31	One or More Eligible [%]	31.26



Section Body
provides additional details (tables & charts)

Section Config
completely data driven add/modify/change sections

NSIGHT COMPUTE

Unguided Analysis / Rules System

Analysis Rules
recommendations from nvvp and more

Recommendations

Bottleneck [Warning] Memory is more heavily utilized than Compute: Look at the Memory Workload Analysis report section to see where the memory system bottleneck is. Check memory replay (coalescing) metrics to make sure you're efficiently utilizing the bytes transferred. Also consider whether it is possible to do more work per memory access (kernel fusion) or whether there are values you can (re)compute.

Roofline Analysis The ratio of peak float (fp32) to double (fp64) performance on this device is 32:1. The kernel achieved 19% of this device's fp32 peak performance and 0% of its fp64 peak performance.

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.

Executed Ipc Elapsed [inst/cycle]	1.17	(-48.02%)	SM Busy [%]	38.29	(-35.00%)
Executed Ipc Active [inst/cycle]	1.33	(-43.60%)	Issue Slots Busy [%]	33.24	(-43.59%)
Issued Ipc Active [inst/cycle]	1.33	(-43.59%)	-		-

Memory Workload Analysis

Detailed analysis of the memory resources of the GPU. Memory can become a limiting factor for the overall kernel performance when fully ng the available communication bandwidth between those units (Max Bandwidth), or by reaching the maximum throughput of issuing memory instructions (ailed tables with data for each memory unit. Deprecated UI elements for backwards compatibility.

Rules Config
completely data driven add/modify/change rules

NSIGHT COMPUTE

Diff'ing kernel runs

Baseline
from any previous profile report
(different kernel, gpu, …)

Metric delta
current values and changes
from baseline

Chart difference
current values and
baseline values

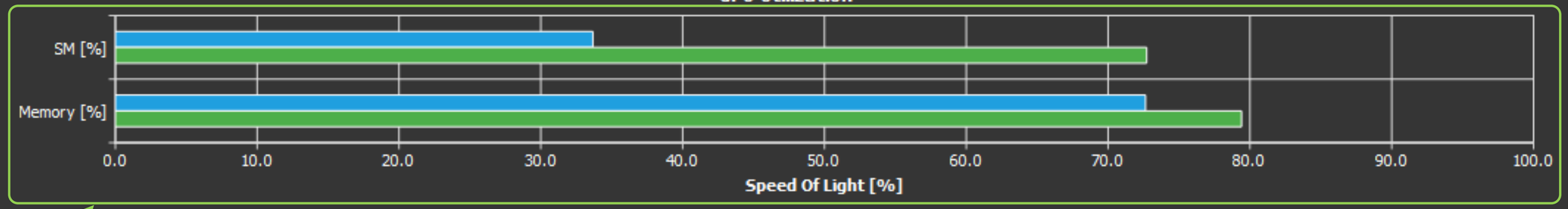
GPU Speed Of L

All

High-level overview o U. For each unit, the Speed Of Light (SOL) reports the achieved percentage of utilization with respect to the theoretical maximum. High-level overview of the utilization for compute and memory e GPU presented as a roofline chart.

SOL SM [%]	33.67	(-53.70%)	Duration [usecond]	77.47	(-55.65%)
SOL Memory [%]	72.64	(-8.52%)	Elapsed Cycles [cycle]	109,970	(-29.67%)
SOL L1/TEX Cache [%]	16.34	(-80.37%)	SM Active Cycles [cycle]	95,946.67	(-35.60%)
SOL L2 Cache [%]	26.76	(+66.22%)	SM Frequency [cycle/nsecond]	1.41	(+57.53%)
SOL DRAM [%]	72.64	(+5,888.33%)	DRAM Frequency [cycle/nsecond]	1.82	(+50.84%)

GPU Utilization



SOL SM Breakdown

	33.67 (+19.99%)
	29.22 (-48.01%)
	29.21 (-48.02%)
	17.20 (+inf%)
d On Any [%]	14.90 (+66.33%)
	8.62 (-70.33%)
SOL SM: Inst Executed Pipe Lsu [%]	7.91 (-89.13%)
SOL SM: Mio Pq Read Cycles Active [%]	4.69 (+6,651.76%)
SOL SM: Mio Pq Write Cycles Active [%]	4.69 (+6,655.45%)
SOL SM: Mio Inst Issued [%]	2.67 (-92.95%)
SOL SM: Mio2rf Writeback Active [%]	1.22 (-96.36%)
SOL SM: Inst Executed Pipe Adu [%]	0.09 (-98.49%)
SOL IDC: Request Cycles Active [%]	0 (+0.00%)
SOL SM: Inst Executed Pipe Fp16 [%]	0 (+0.00%)

SOL Memory Breakdown

SOL GPU: Dram Throughput [%]	72.64 (+5,888.33%)
SOL L2: T Sectors [%]	26.76 (+66.25%)
SOL L2: Lts2xbar Cycles Active [%]	16.10 (+7.34%)
SOL L2: D Sectors Fill Device [%]	16.05 (+2,051.46%)
SOL L2: Xbar2lts Cycles Active [%]	14.41 (-6.68%)
SOL L2: D Sectors [%]	13.33 (+63.38%)
SOL L1: M Xbar2l1tex Read Sectors [%]	9.38 (-15.56%)
SOL L1: M L1tex2xbar Req Cycles Active [%]	8.17 (-10.39%)
SOL L1: Lsuin Requests [%]	7.91 (-89.13%)
SOL L2: T Tag Requests [%]	6.72 (-58.25%)
SOL L1: Data Pipe Lsu Wavefronts [%]	4.28 (-94.61%)
SOL L1: Lsu Writeback Active [%]	2.39 (-96.42%)
SOL L1: Data Bank Reads [%]	1.96 (-76.61%)
SOL L1: Data Bank Writes [%]	1.96 (+4.25%)

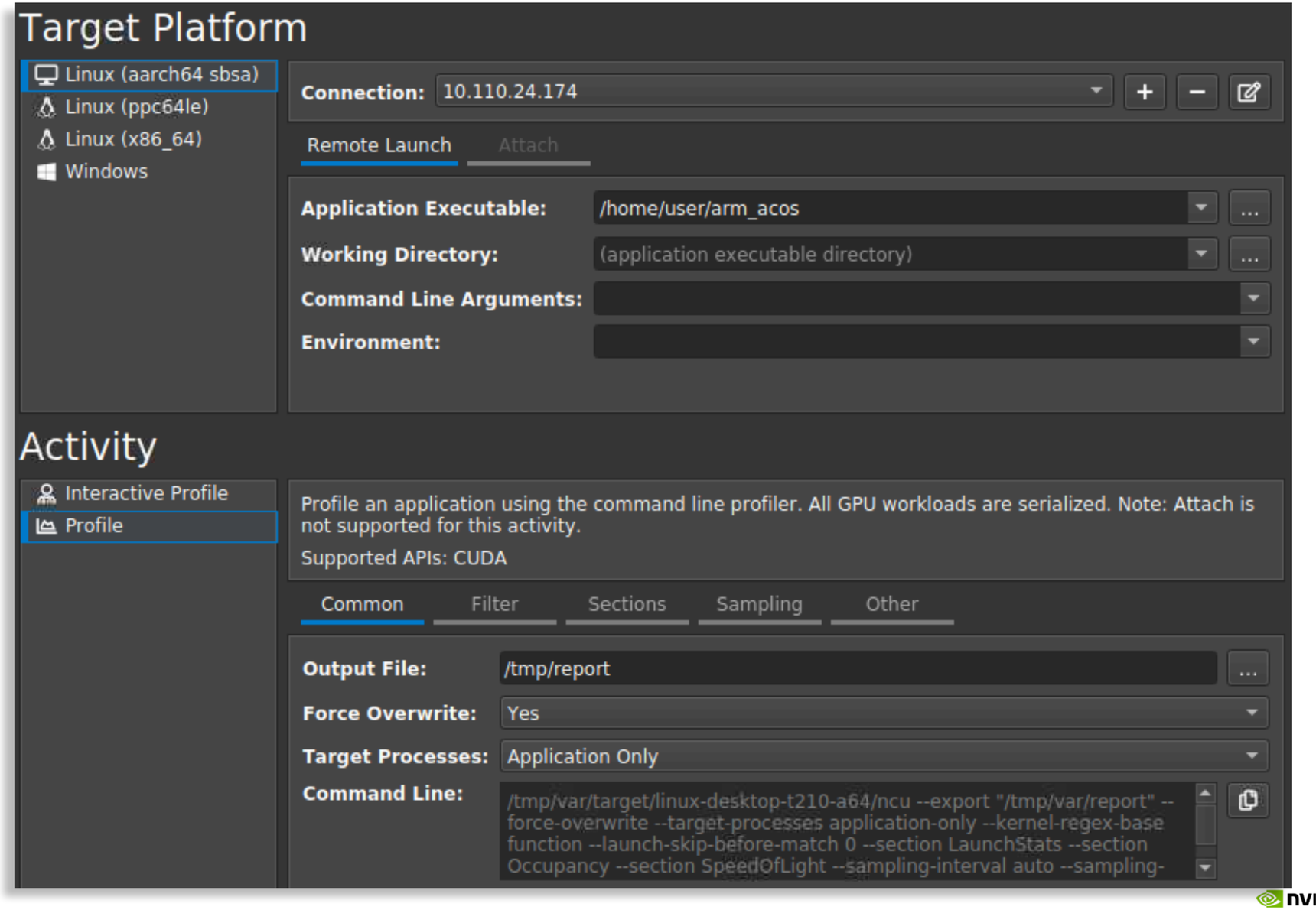
NVIDIA

NSIGHT COMPUTE

Support for Arm (SBSA) target architecture

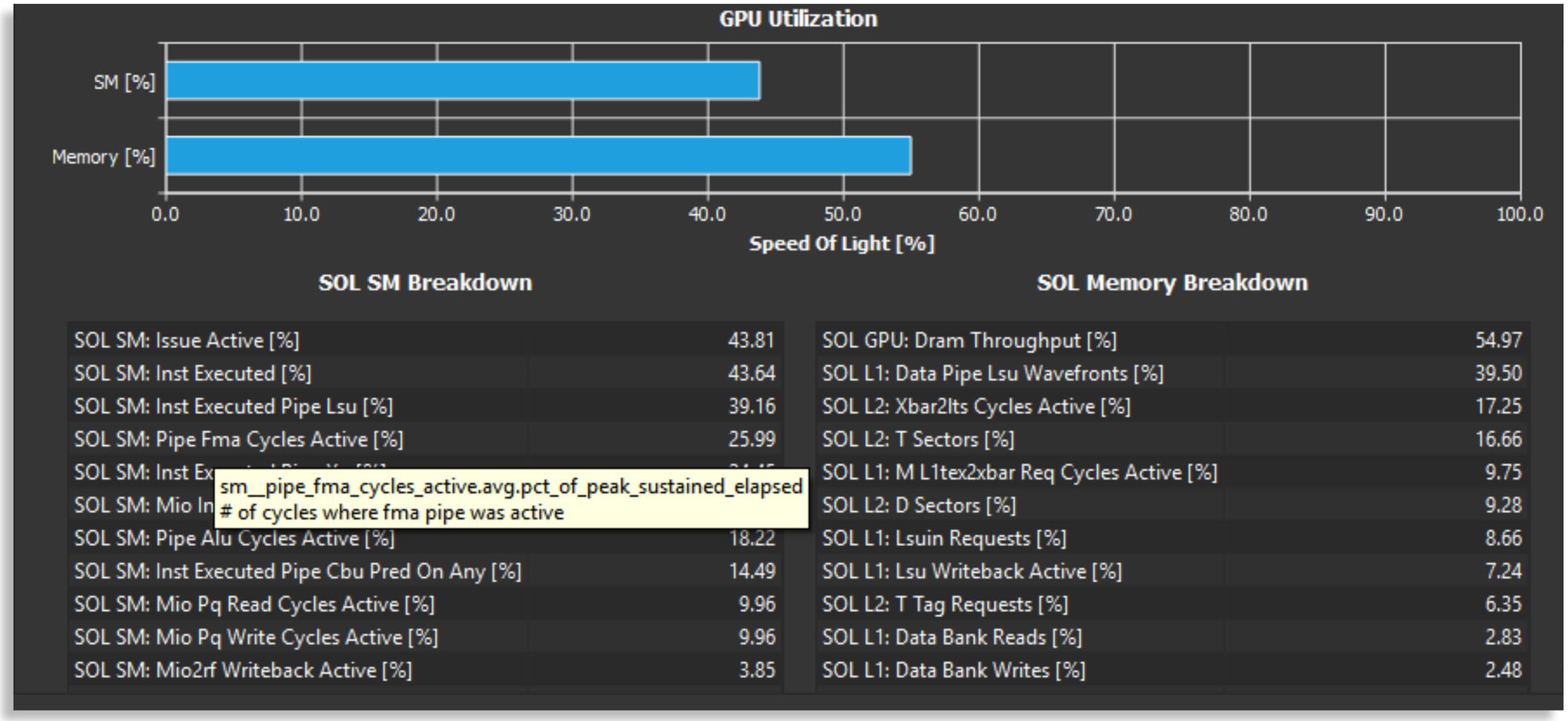
Support for PowerPC target architecture

Profile activity command line is shown



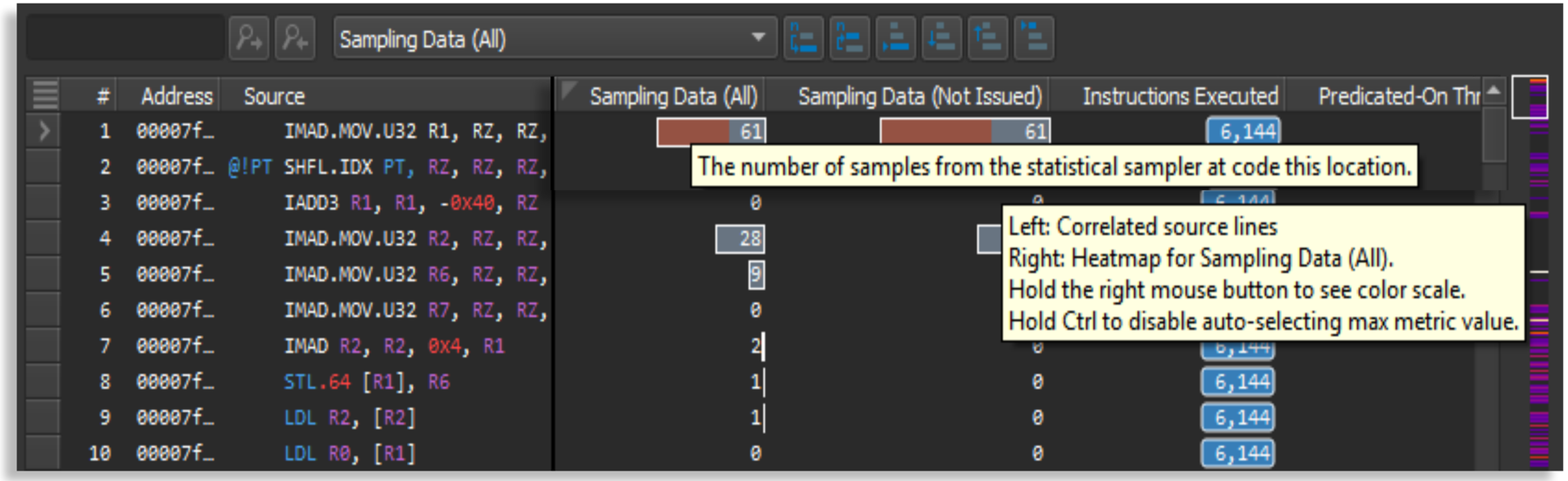
NSIGHT COMPUTE

Detailed breakdown of high-level SOL metrics



NSIGHT COMPUTE

Improved help for Source page and Details page charts



NSIGHT COMPUTE

A100 GPU support

Support for CUDA Asynchronous Copy

Sparse Data compression

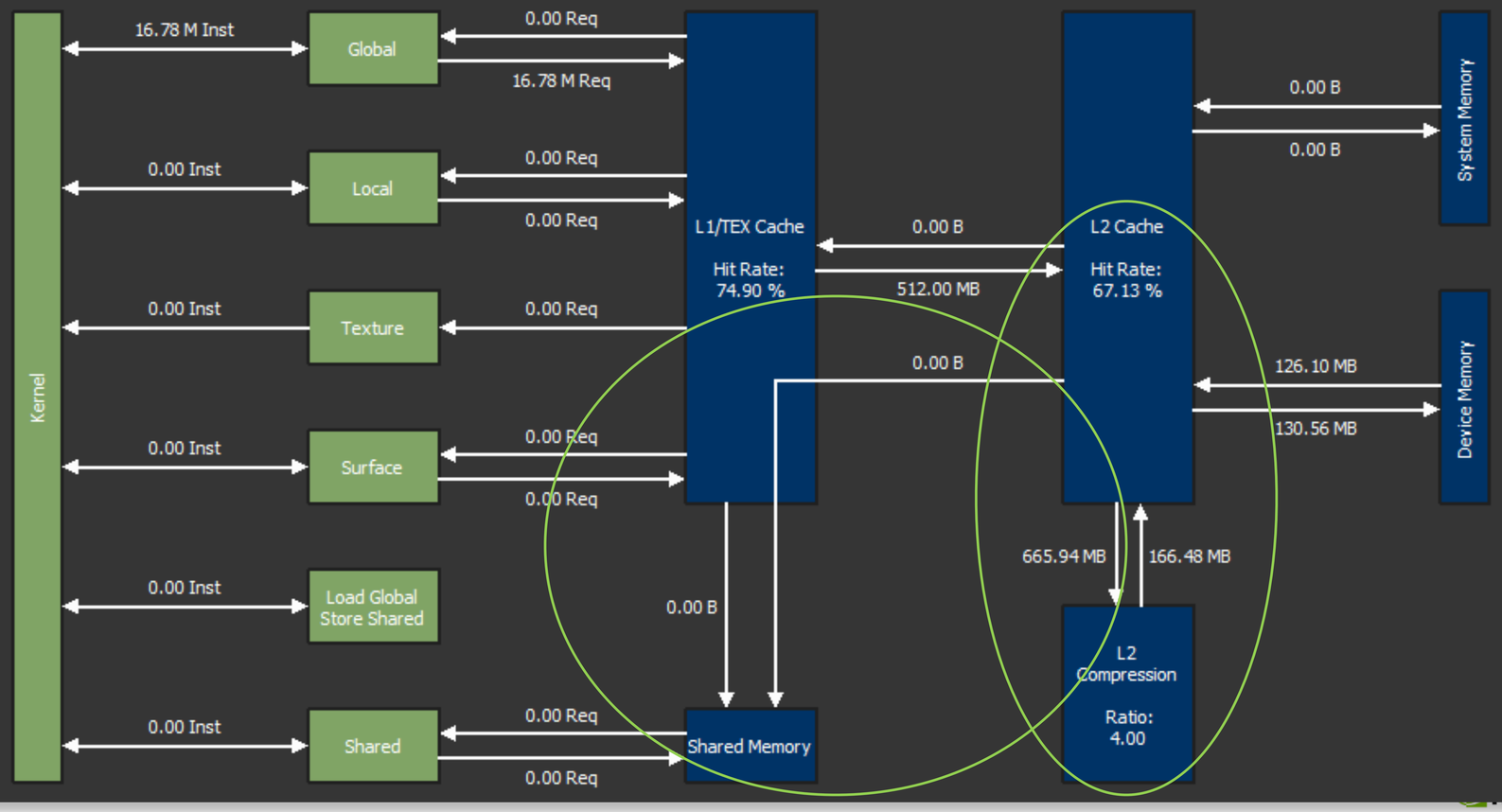
Memory Workload Analysis

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]	564.77	Mem Busy [%]	68.43
L1/TEX Hit Rate [%]	74.90	Max Bandwidth [%]	33.92
L2 Hit Rate [%]	67.13	Mem Pipes Busy [%]	25.34
L2 Compression Success Rate [%]	100	L2 Compression Ratio	4

Memory Chart



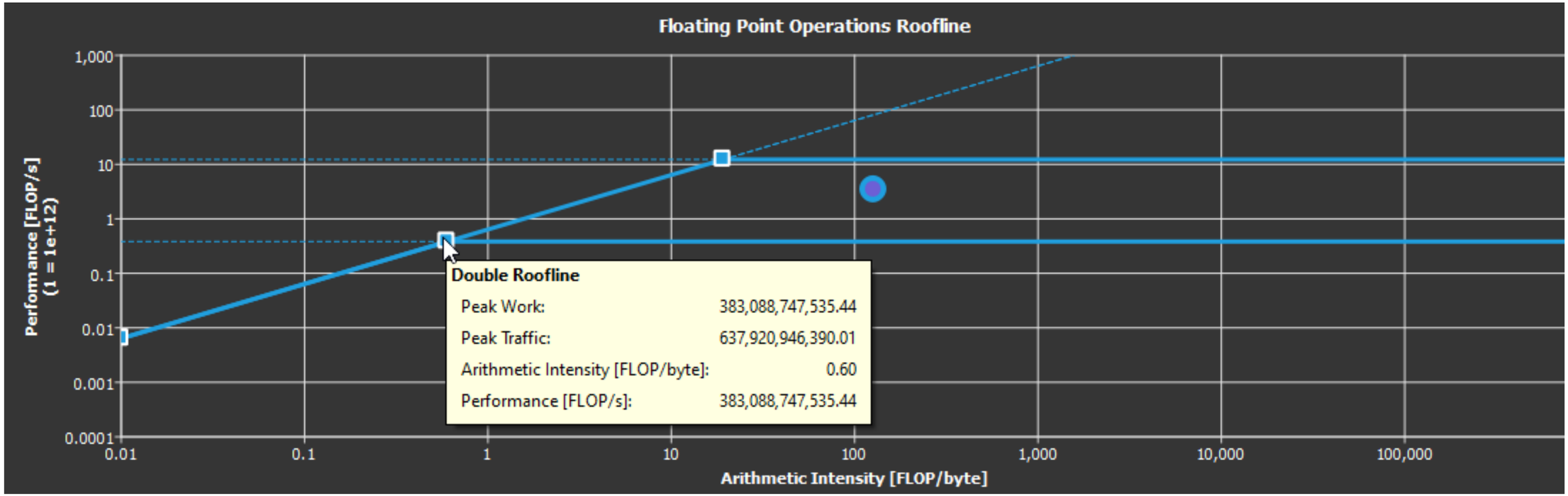
NSIGHT COMPUTE

New Roofline analysis

Efficient way to evaluate kernel characteristics, quickly understand potential directions for further improvements or existing limiters

Inputs: Arithmetic Intensity (FLOPS/bytes)
Performance (FLOPS/s)

Ceilings: Peak Memory Bandwidth
Peak FP32/FP64 Performance



NSIGHT COMPUTE

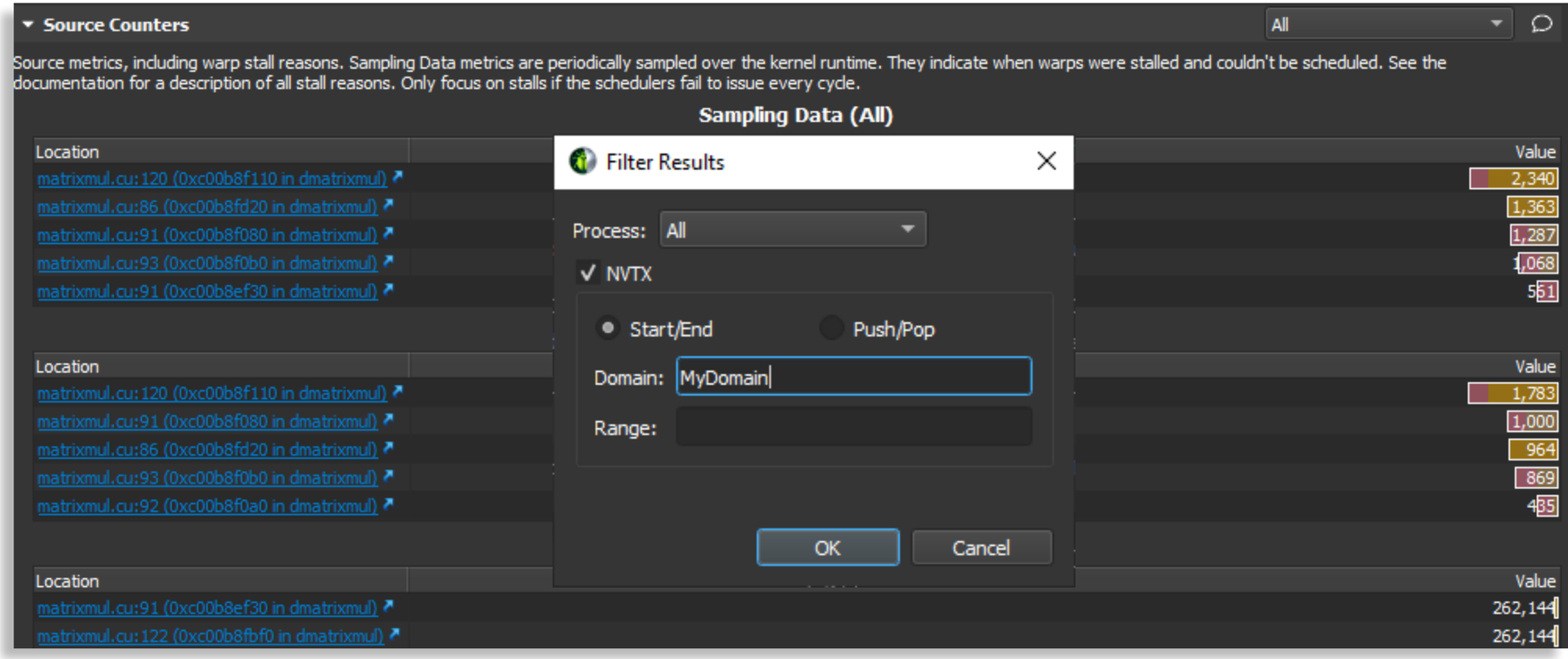
Workflow Improvements

New Hot Spot Tables

Result Filtering by NVTX

Source Page can now be collapsed to generate aggregate metrics

Recommendations: links to other sections & lines on Source page, access instanced metrics



NSIGHT COMPUTE

Other Improvements

Shortened Executables and File Extensions

Versions	UI	CLI	Report file extension
2019.5 (CUDA 10.2) and before	`nv-nsight-cu`	`nv-nsight-cu-cli`	`.nsight-cuprof-report`
2020.1 (CUDA 11.0) and newer	`ncu-ui`	`ncu`	`.ncu-rep`

Ol

New rule for re

Support report name placeholders `%p`, `%q`, `%i`, and `%h`

Improved documentation

AGENDA

- ❖ Using Nsight Compute profiler
- ➢ Intro to Profile Devito generated Code with Nsight Compute
- ❖ Exercise 1: Coalesced Memory
- ❖ Exercise 2: Block Size
- ❖ Exercise 3: Reduce Memory Reads
- ❖ Exercise 4 : Maximize cache hits
- ❖ Exercise 5: Prefetch with Shared Memory

RUNNING NSIGHT COMPUTE

- Command-line:
  - `ncu [options] program [program arguments]`
- Command-line options:
  - `-s #` : skip a number of kernels from the beginning
  - `-c #` : profile a number of kernels
  - `-f` : force output overwrite
  - `-o filename` : output results to filename.ncu-rep for analysis on any other system where nsight compute is installed
  - `--section "regex:.*"` : profile for all sections satisfying the regular expression
- Example:
  - `ncu -c 3 -f -o dummy_benchmark --section "regex:.*" python run.py -d 512 512 512 -nt 10 --broken-mpi True`
  - Will:
    - Profile 3 kernels
    - Output to dummy_benchmark.ncu-rep
    - Generate all sections
    - Execute the devito example run.py with 512x512x512 configuration for 10 timesteps without MPI
- Note:
  - *Previous ncu versions will have a different section understanding: --section ".*" shall then work*

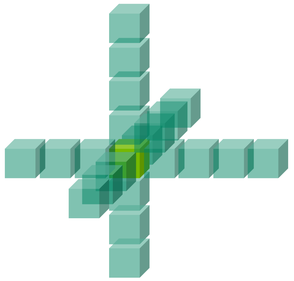
AGENDA

- ❖ Using Nsight Compute profiler
- ❖ Intro to Profile Devito generated Code with Nsight Compute
- ➢ Exercise 1: Coalesced Memory
- ❖ Exercise 2: Block Size
- ❖ Exercise 3: Reduce Memory Reads
- ❖ Exercise 4 : Maximize cache hits
- ❖ Exercise 5: Prefetch with Shared Memory

OVERVIEW

Mainly two kernels [Section 1]

- Kernel 1
  - STENCIL on p field [-4..+3] in each direction



  - Use 6 other fields
  - Output of 3 fields r12,r13,r14

- Kernel 2
  - STENCIL on r12,r13,r14 fields [-4..+3] one direction per field
  - Use 3 other fields and previous values of p
  - Output to p

- Domain size: 512x512x512 – r size: 519^3 – p size : ~526^3
  - P + other fields : 582 + 559*6 = 3.94GB reads
  - R output : 1.68GB writes

- Domain size: 512x512x512 – r size: 519x512x512
  - R + other fields : 1633 + 537*3 = 3.24GB reads
  - P output : 537MB writes

THEORETICAL SPEED OF LIGHT (minimal amount of data to Read/Write)

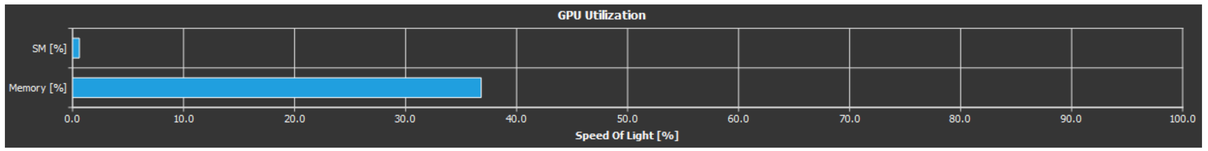
3.94 GB read / 1.68 GB write

3.24 GB read / 0.54 GB write

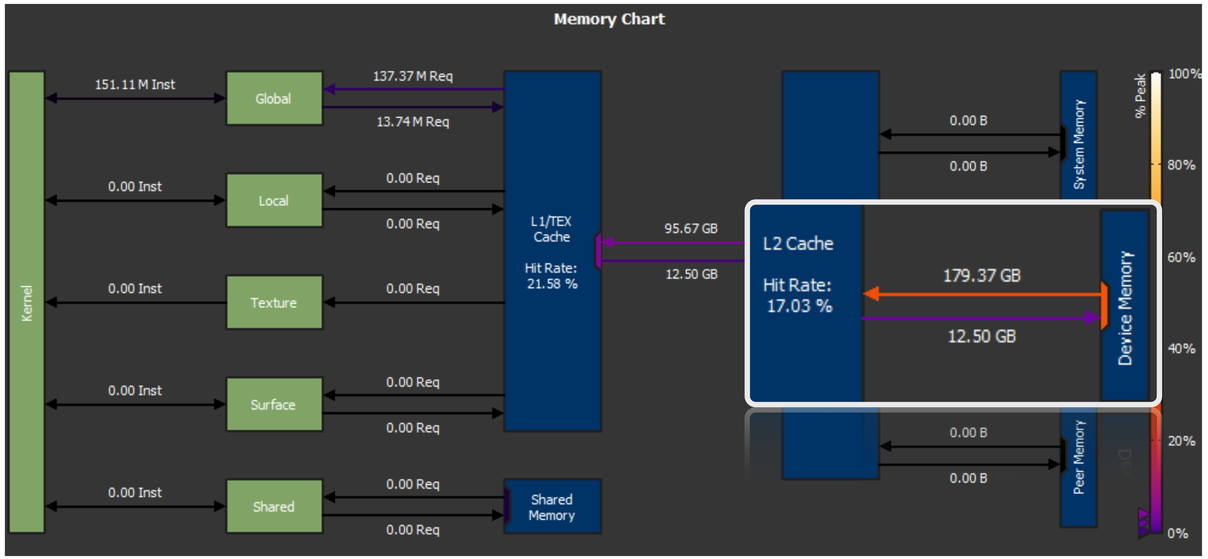
EXERCISE 1: COALESCED MEMORY

Devito Generated Kernels

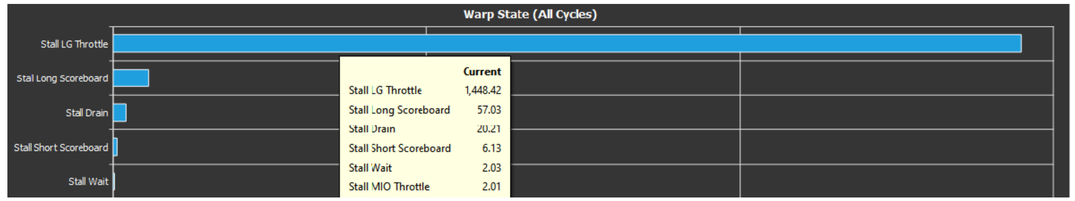
- Kernel 1
  - 36 % of memory SOL



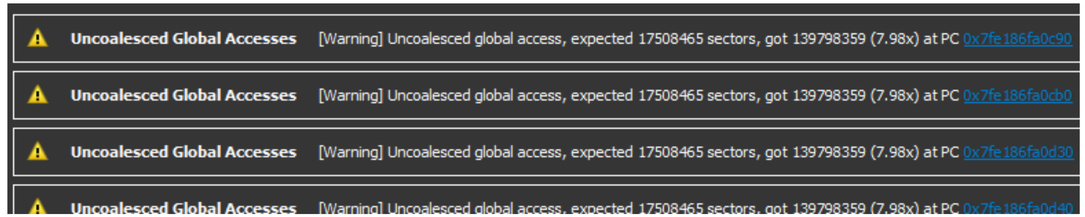
  - ~180 GB of reads



- Symptoms
  - Higher than necessary memory read (180GB instead of ~4GB)
  - Low memory speed of light
  - High LG Throttle as Stall Reason (Warp State)



  - Source counters section warnings



THEORETICAL SPEED OF LIGHT (minimal amount of data to Read/Write)

3.94 GB read / 1.68 GB write

EXERCISE 1: COALESCED MEMORY

Devito Generated Kernels

```
#define r0L0(x, y, z) r0[(x)*y_stride1 + (y)*z_stride1 + (z)]
```

- Uncoalesced

```
int x = -4 + (int)(blockIdx.x*blockDim.x + threadIdx.x);
int y = -4 + (int)(blockIdx.y*blockDim.y + threadIdx.y);
int z = -4 + (int)(blockIdx.z*blockDim.z + threadIdx.z);
```

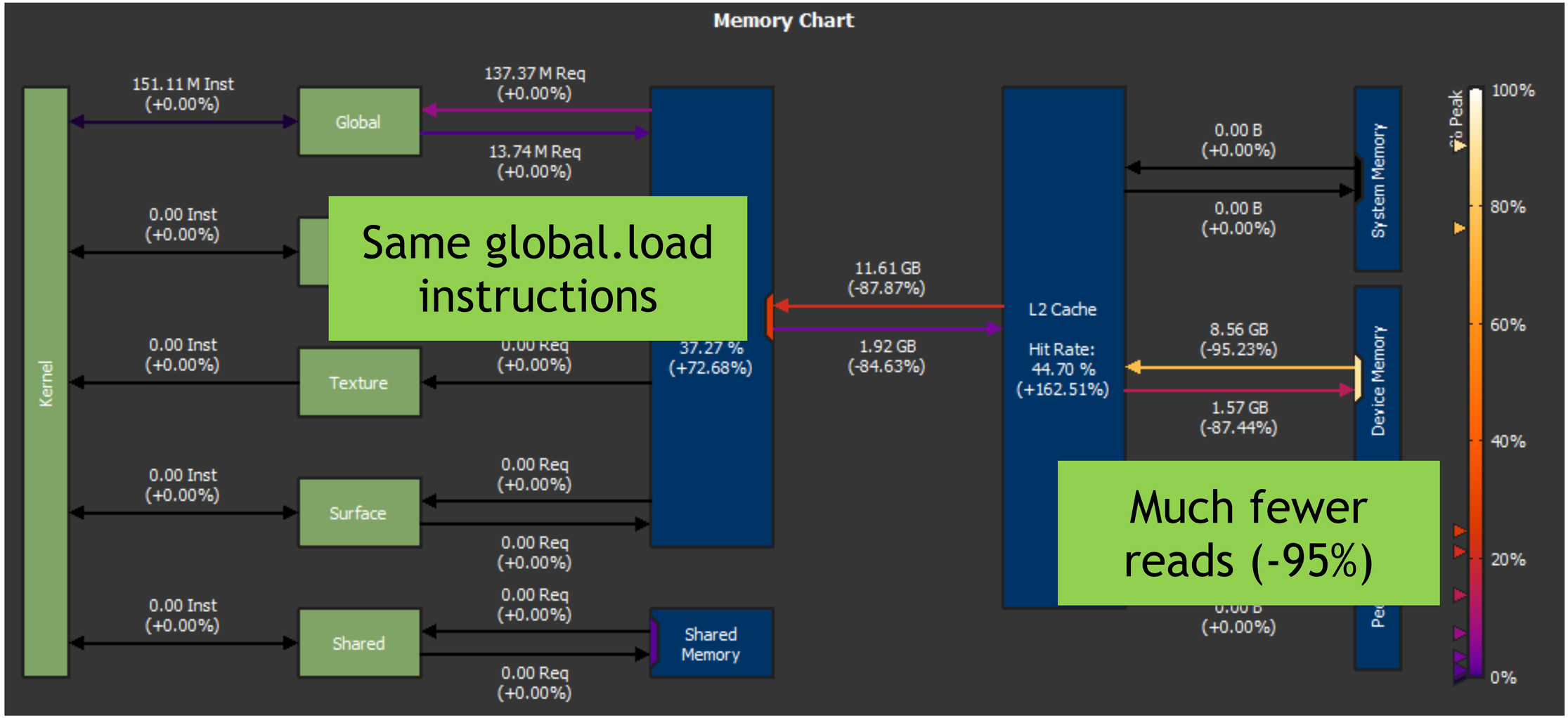
- X index (highest stride) on threadIdx.x

- Coalesced

```
int z = -4 + (int)(blockIdx.x*blockDim.x + threadIdx.x);
int y = -4 + (int)(blockIdx.y*blockDim.y + threadIdx.y);
int x = -4 + (int)(blockIdx.z*blockDim.z + threadIdx.z);
```

- Z index (unit stride) on threadIdx.x

EXERCISE 1: COALESCED MEMORY



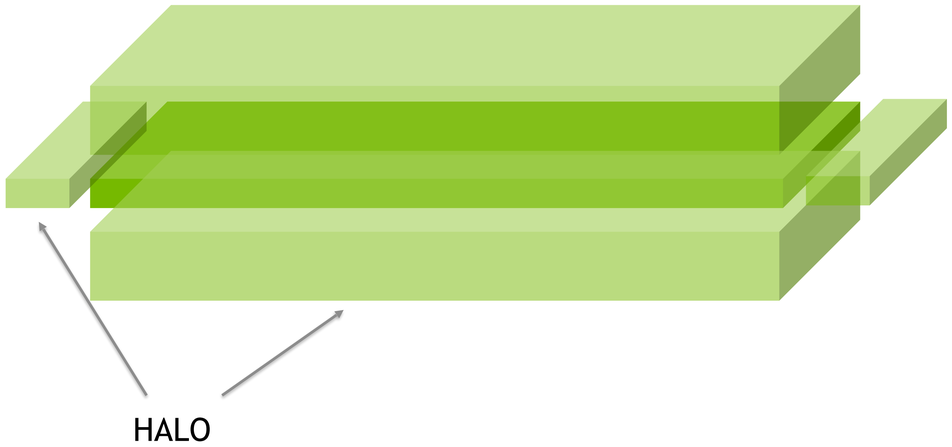
AGENDA

- Using Nsight Compute profiler
- Intro to Profile Devito generated Code with Nsight Compute
- Exercise 1: Coalesced Memory
- ➢ Exercise 2: Block Size
- Exercise 3: Reduce Memory Reads
- Exercise 4 : Maximize cache hits
- Exercise 5: Prefetch with Shared Memory

EXERCISE 2: MAXIMIZE BLOCK SIZE

Improve L1 cache usage

- Kernel uses less than 64 registers
  - GPU has 65536 registers per SM
- Current version
  - blockDim: 32,4,1 = 256 threads



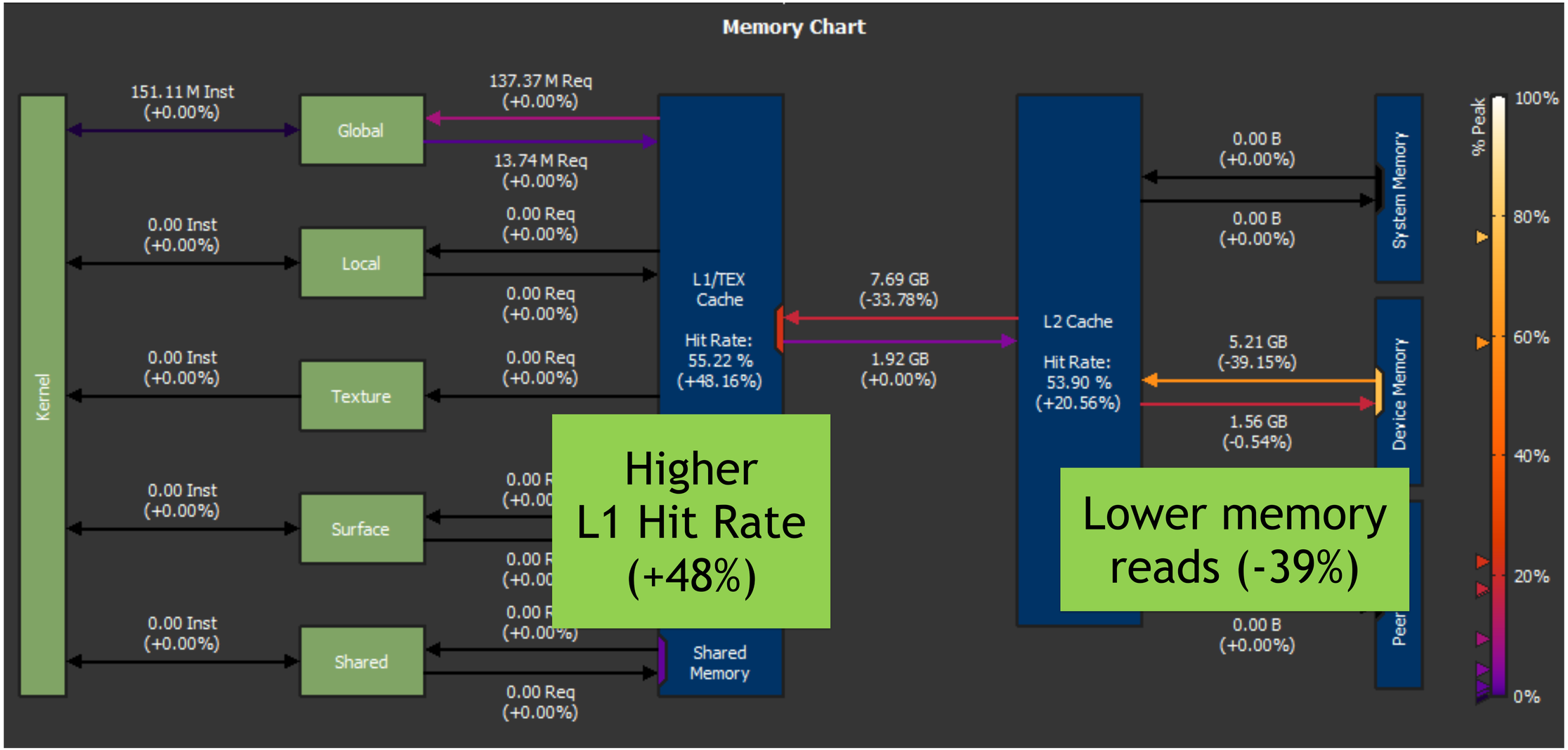
- Halo size : 7*4*1 + 7*32*4 + 7*32*1 = 1148
- Inner size: 32*4*1 = 128
- Overhead: 10x

- Proposed version
  - blockDim: 32,4,8 = 1024 threads
-
  -

- Halo size: 7*4*8 + 7*32*4 + 7*32*8 = 2912
- Inner size: 32*4*8 = 1024
- Overhead: 4x

EXERCISE 2: MAXIMIZE BLOCK SIZE

Improve L1 cache usage



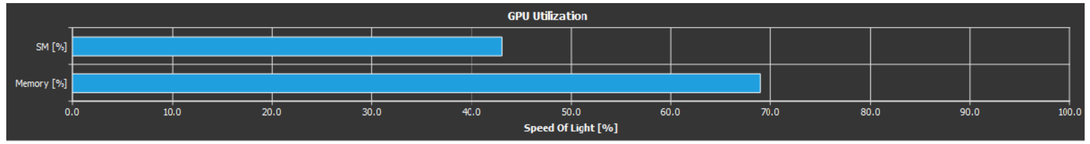
PERFORMANCE MILESTONE

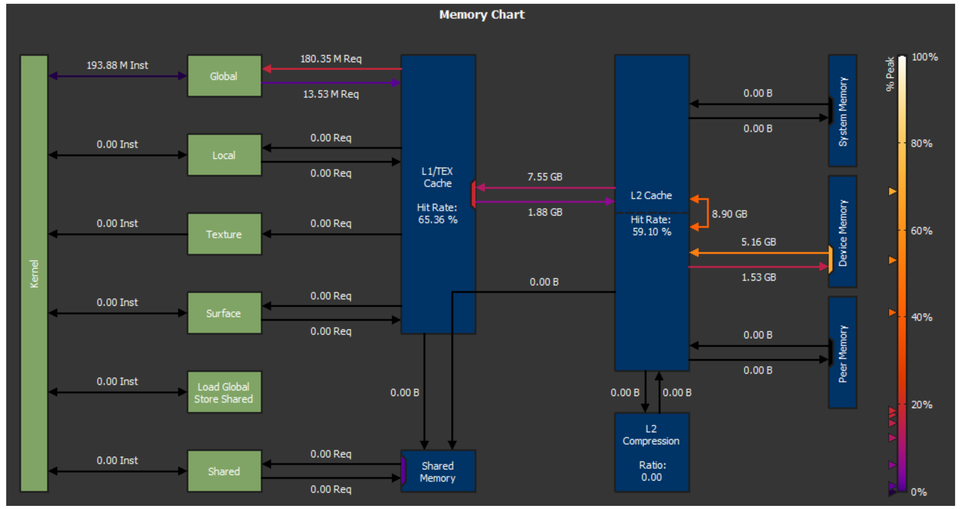
2 Main kernels – [Section 1]

- Kernel 1 – 6.59ms
  - Work distribution – 32x4x8 threads per block
  - Round-robin on blocks

```
for (int x = from.x + threadIdx.z + blockIdx.z * blockDim.z ; x <= to.x ; x += blockDim.z * gridDim.z)
for (int y = from.y + threadIdx.y + blockIdx.y * blockDim.y ; y <= to.y ; y += blockDim.y * gridDim.y)
for (int z = from.z + threadIdx.x + blockIdx.x * blockDim.x ; z <= to.z ; z += blockDim.x * gridDim.x)
```

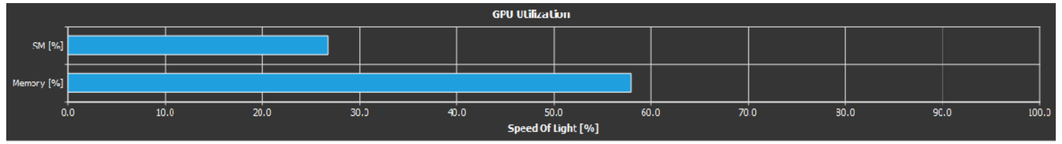
  - Plain access to arrays

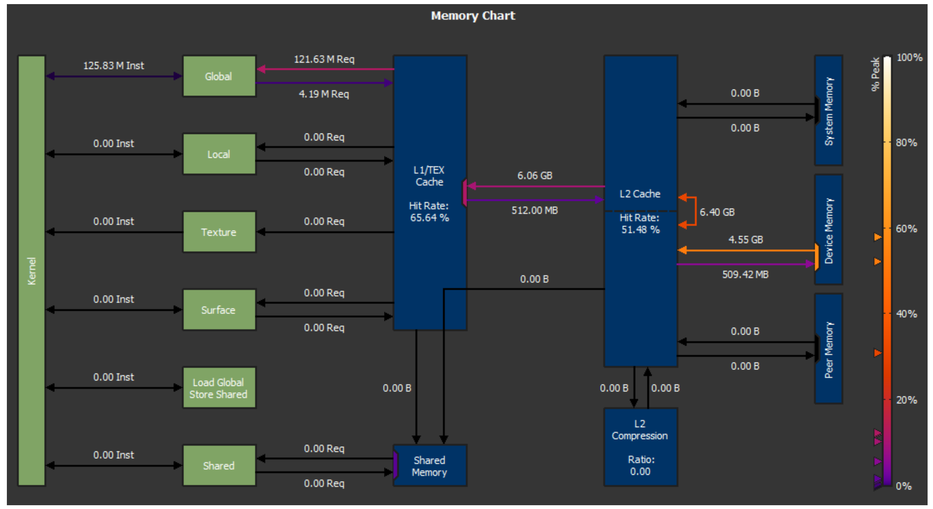




3.94 GB read / 1.68 GB write

- Kernel 2 – 6.01ms
  - Same work distribution





3.24 GB read / 0.54 GB write

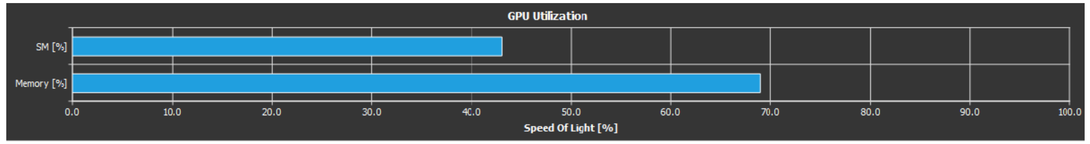
PERFORMANCE MILESTONE

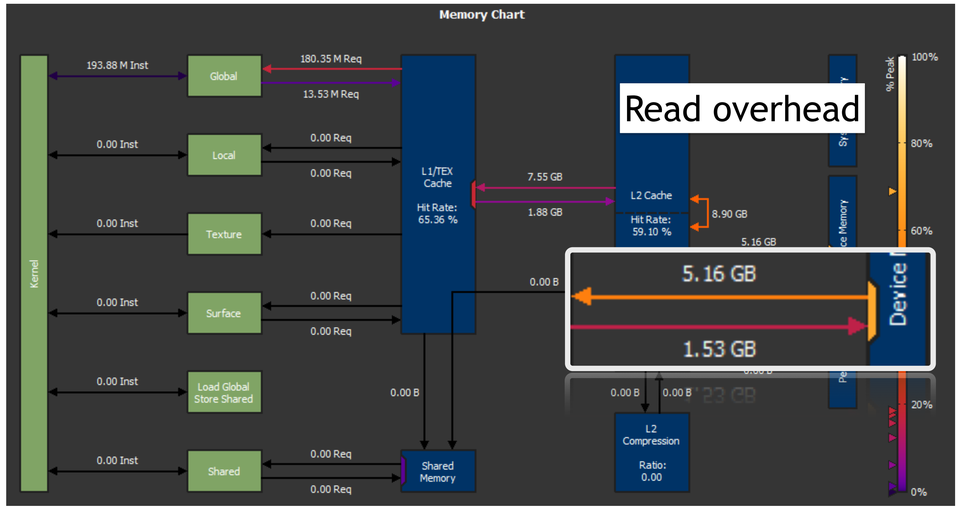
2 Main kernels – [Section 1]

- Kernel 1 – 6.59ms
  - Work distribution – 32x4x8 threads per block
  - Round-robin on blocks

```
for (int x = from.x + threadIdx.z + blockIdx.z * blockDim.z ; x <= to.x ; x += blockDim.z * gridDim.z)
for (int y = from.y + threadIdx.y + blockIdx.y * blockDim.y ; y <= to.y ; y += blockDim.y * gridDim.y)
for (int z = from.z + threadIdx.x + blockIdx.x * blockDim.x ; z <= to.z ; z += blockDim.x * gridDim.x)
```

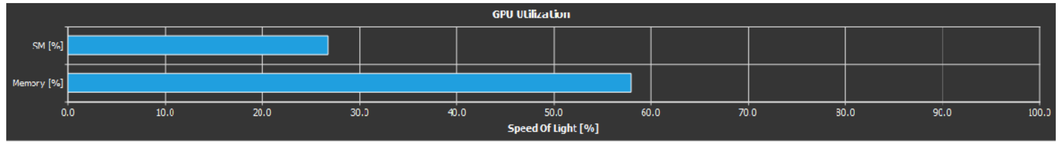
  - Plain access to arrays

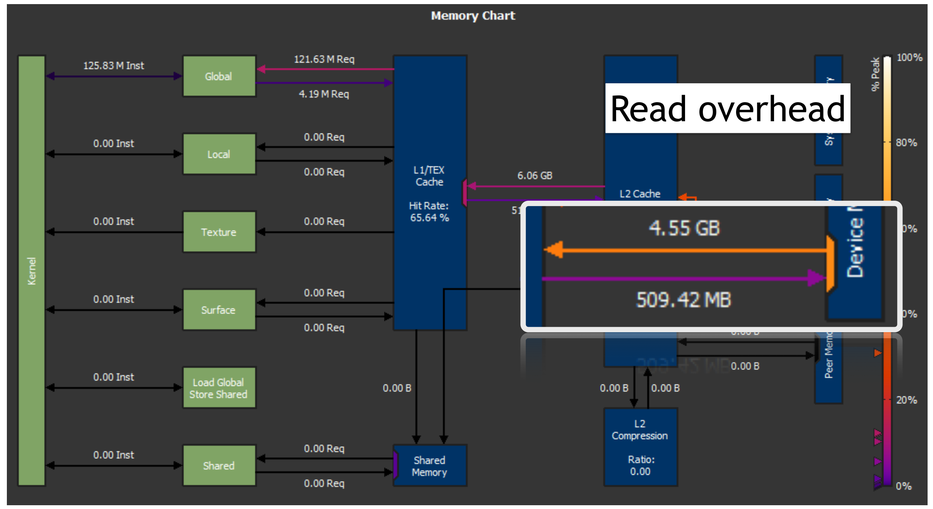




3.94 GB read / 1.68 GB write

- Kernel 1 – 6.01ms
  - Same work distribution





3.24 GB read / 0.54 GB write

AGENDA

- Using Nsight Compute profiler
- Intro to Profile Devito generated Code with Nsight Compute
- Exercise 1: Coalesced Memory
- Exercise 2: Block Size
- ➢ Exercise 3: Reduce Memory Reads
- Exercise 4 : Maximize cache hits
- Exercise 5: Prefetch with Shared Memory

EXERCISE 3 – REDUCE MEMORY READS

Remove sin/cos pre-calculation

- EXERCISE
  - Move sin and cos from kernel_0 to calculations in kernel_1_part_0

```
        r0(x+4,y+4,z+4) = ::cosf(theta(x+8,y+8,z+8)) ;
        r1(x+4,y+4,z+4) = ::cosf(phi(x+8,y+8,z+8)) ;
        r2(x+4,y+4,z+4) = ::sinf(theta(x+8,y+8,z+8)) ;
        r3(x+4,y+4,z+4) = ::sinf(phi(x+8,y+8,z+8)) ;
```

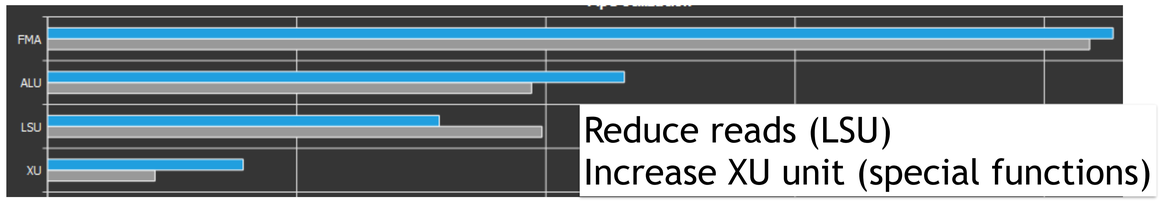
- Goal: sin and cos are calculated on the fly

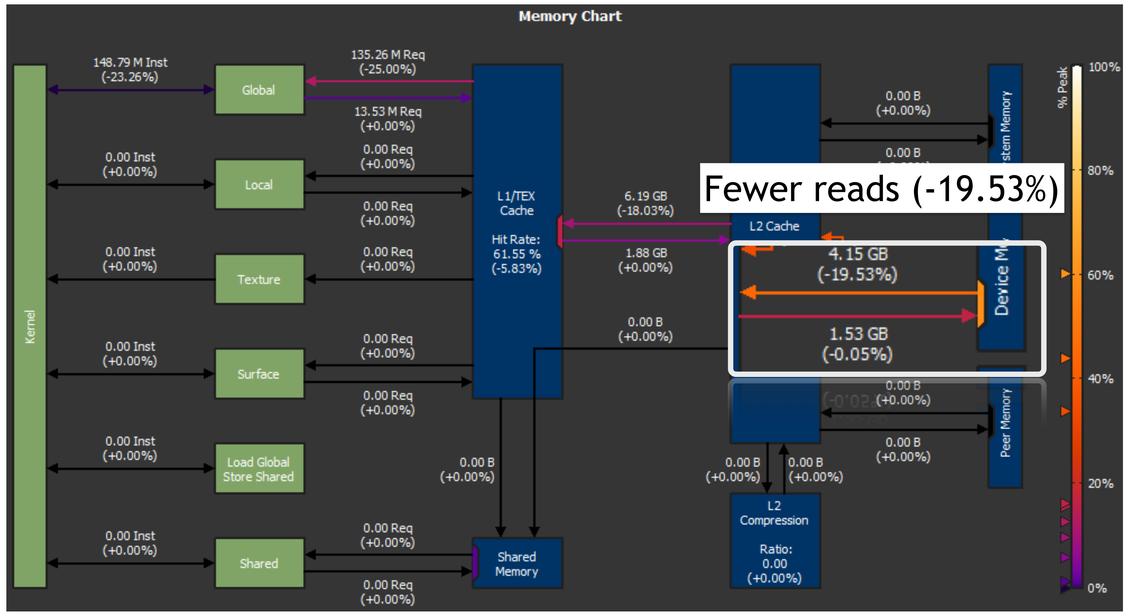
```
        int jndex = thetaL0.index(x + 8, y + 8, z + 8);

        float sintheta, costheta, sinphi, cosphi ;
        ::sincosf(thetaL0.get<>(jndex), &sintheta,
&costheta);
        ::sincosf(phiL0.get<>(jndex), &sinphi, &cosphi);
```

EXERCISE 3 – REDUCE MEMORY READS

Remove sin/cos pre-calculation

- Comparing kernel_1_part_0 in NSIGHT Compute
- 6.59ms -> 6.52ms





3.94 GB read / 1.68 GB write

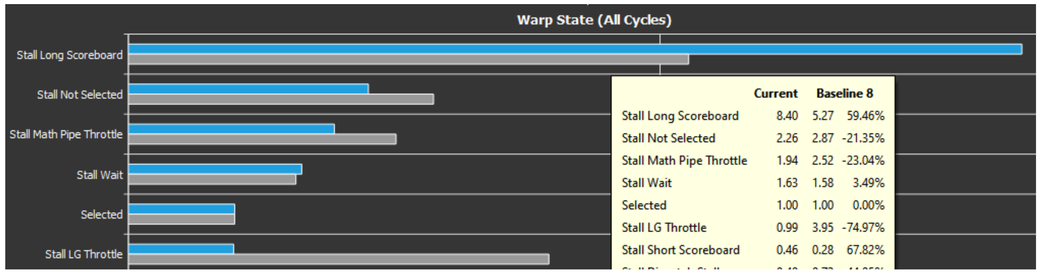
- Before: r0,r1,r2,r3 used as buffers initialized by kernel_section_0

```
r0(x+4,y+4,z+4) = ::cosf(theta(x+8,y+8,z+8)) ;
r1(x+4,y+4,z+4) = ::cosf(phi(x+8,y+8,z+8)) ;
r2(x+4,y+4,z+4) = ::sinf(theta(x+8,y+8,z+8)) ;
r3(x+4,y+4,z+4) = ::sinf(phi(x+8,y+8,z+8)) ;
```

- After: sin and cos are calculated on the fly

```
int jndex = thetaL0.index(x + 8, y + 8, z + 8);

float sintheta, costheta, sinphi, cosphi ;
::sincosf(thetaL0.get<>(jndex), &sintheta,
&costheta);
::sincosf(phiL0.get<>(jndex), &sinphi, &cosphi);
```

- Though, higher latency to load data here



AGENDA

- ❖ Using Nsight Compute profiler
- ❖ Intro to Profile Devito generated Code with Nsight Compute
- ❖ Exercise 1: Coalesced Memory
- ❖ Exercise 2: Block Size
- ❖ Exercise 3: Reduce Memory Reads
- ➢ Exercise 4 : Maximize cache hits
- ❖ Exercise 5: Prefetch with Shared Memory

EXERCISE 4 – MAXIMIZE CACHE HITS

Part 1: Change block shape

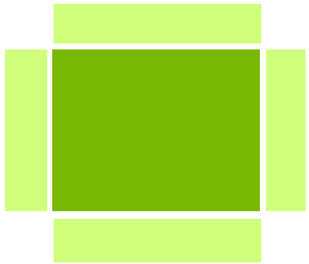
- EXERCISE
  - Change the block dimension towards larger chunk of work per SM

- Before:

```
dim3
block(32,4,8);
```

- We want to reduce the relative size of ghosts vs inner part
- After:

```
dim3 block(32,32,1);
```



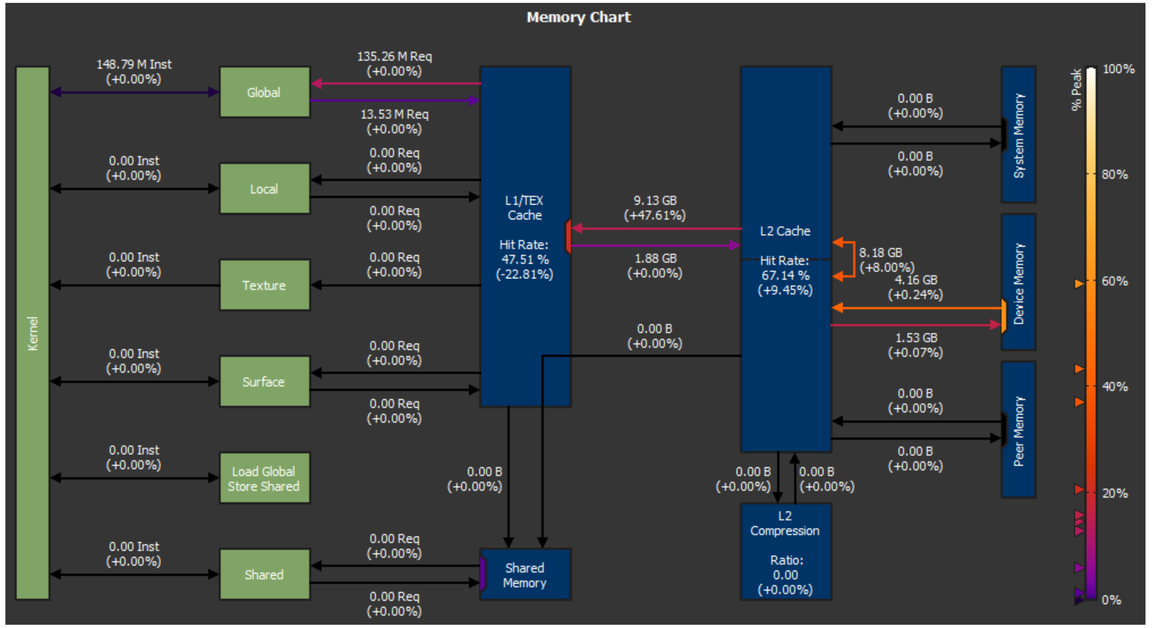
- First step is to have a larger 2D section before iterating in the 3D dimension

3.94 GB read / 1.68 GB write

EXERCISE 4 – MAXIMIZE CACHE HITS

Part 1: Change block shape

- Changing block shape improves performance, but changes cache behavior: L1 hit decreases (23%), L2 hit increases, overall reads similar
- 6.52ms -> 6.50ms



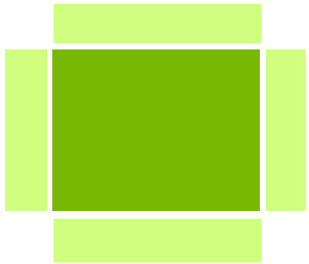
3.94 GB read / 1.68 GB write

- Before:

```
dim3
block(32,4,8);
```

- We want to reduce the relative size of ghosts vs inner part
- After:

```
dim3 block(32,32,1);
```



- First step is to have a larger 2D section before iterating in the 3D dimension

EXERCISE 4 – MAXIMIZE CACHE HITS

Part 2: Each block will operate on a range of X (instead of 1)
Blocks will process 32x32x32 instead of 32x4x8 initially

- Add an inner loop on x to cover multiple X indices per SM

- Code modification
  - Kernel

```
for (int x = cx ; x <= min(to.x, cx+Xchunk-1) ; ++x)
{
```

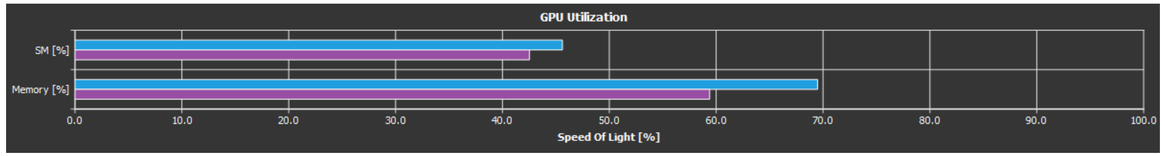
  - Launch configuration

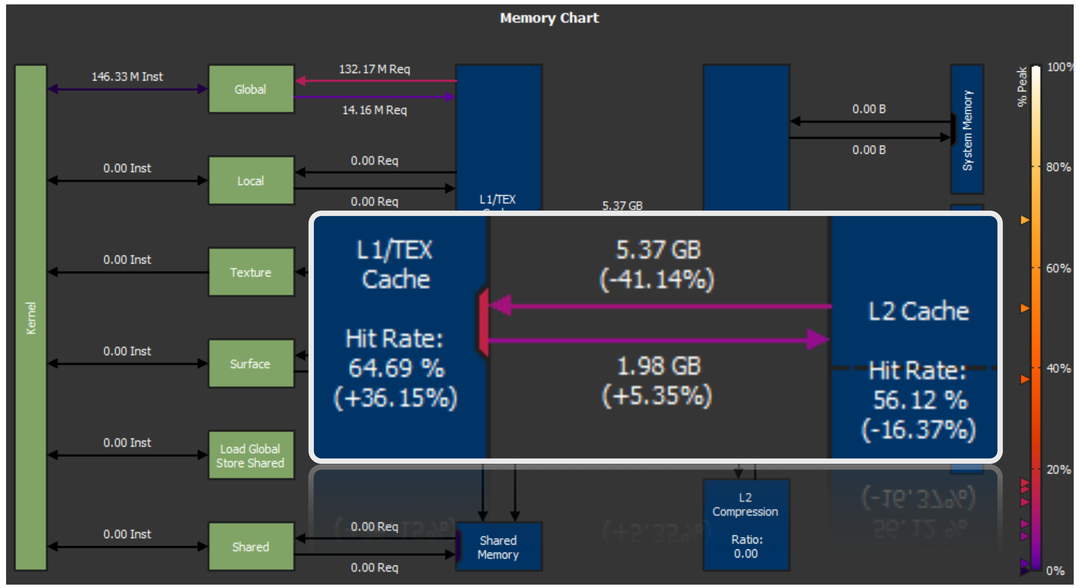
```
dim3 block(32,32,1);
dim3 grid(
    (to.z - from.z + 1 + block.x - 1) / block.x,
    (to.y - from.y + 1 + block.y - 1) / block.y,
    (to.x - from.x + 1 + Xchunk - 1) / Xchunk
) ;
```

EXERCISE 4 – MAXIMIZE CACHE HITS

Part 2: Each block will operate on a range of X (instead of 1)
Blocks will process 32x32x32 instead of 32x4x8 initially

- kernel 0 – 6.50ms -> 6.27ms
- Better Memory subsystem utilization





3.94 GB read / 1.68 GB write

- Code modification
  - Kernel

```
for (int cx = from.x + blockIdx.z * Xchunk ; cx <= (int)to.x ; cx += gridDim.z * Xchunk)
for (int y = from.y + threadIdx.y + blockIdx.y * blockDim.y ; y <= (int)to.y ; y += blockDim.y * gridDim.y)
for (int z = from.z + threadIdx.x + blockIdx.x * blockDim.x ; z <= (int)to.z ; z += blockDim.x * gridDim.x)
{
    for (int x = cx ; x <= min(to.x, cx+Xchunk-1) ; ++x)
    {
```

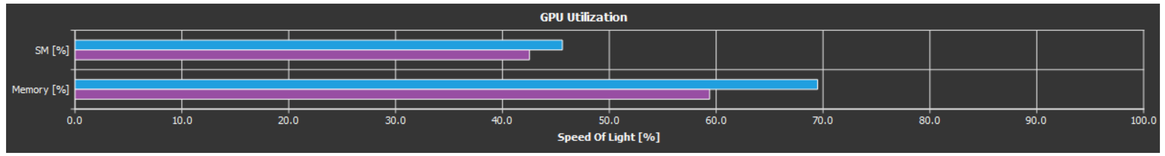
  - Launch configuration

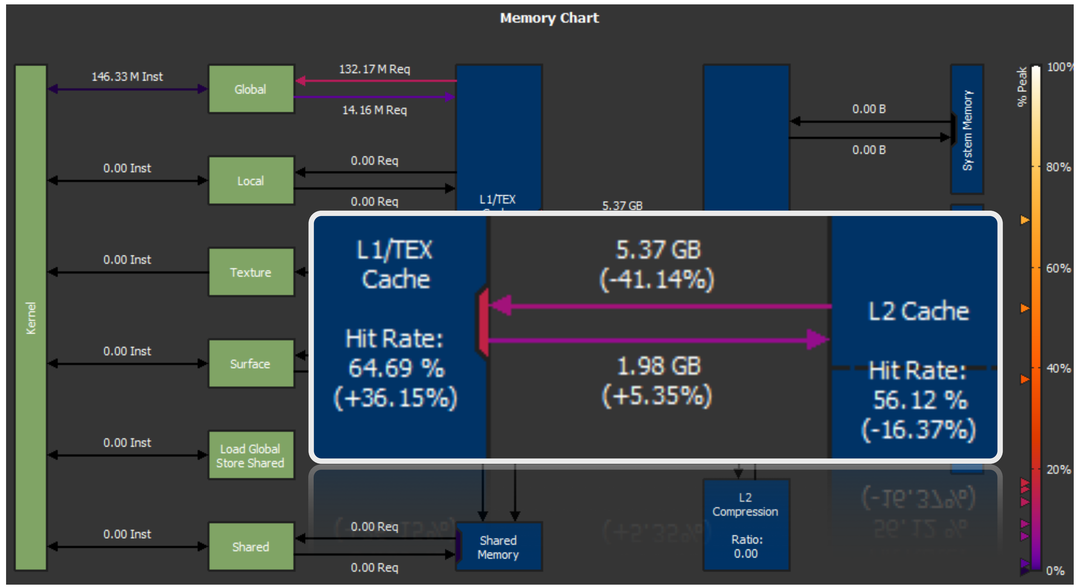
```
dim3 block(32,32,1);
dim3 grid(
    (to.z - from.z + 1 + block.x - 1) / block.x,
    (to.y - from.y + 1 + block.y - 1) / block.y,
    (to.x - from.x + 1 + Xchunk - 1) / Xchunk
) ;
```

EXERCISE 4 – MAXIMIZE CACHE HITS

Part 2: Each block will operate on a range of X (instead of 1)
Blocks will process 32x32x32 instead of 32x4x8 initially

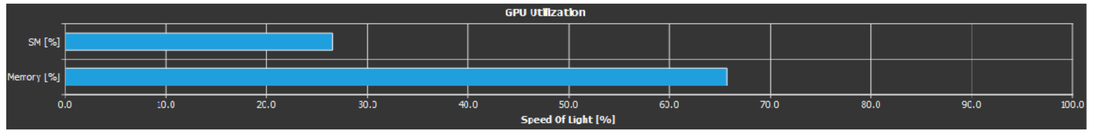
- kernel 0 – 6.50ms -> 6.27ms
- Better Memory subsystem utilization

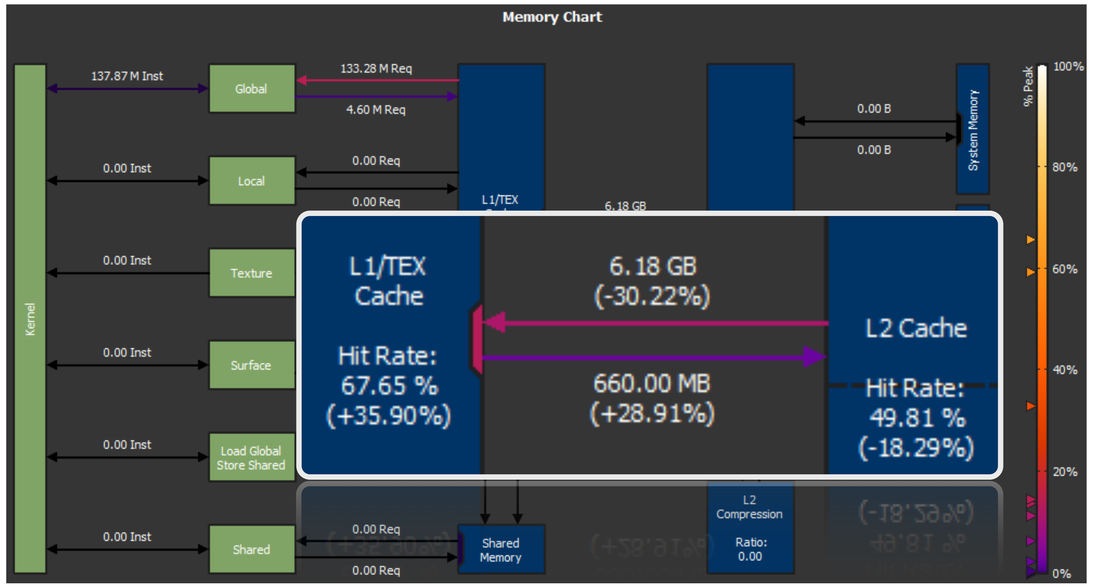




3.94 GB read / 1.68 GB write

- Kernel 1 – 5.52ms





3.24 GB read / 0.54 GB write

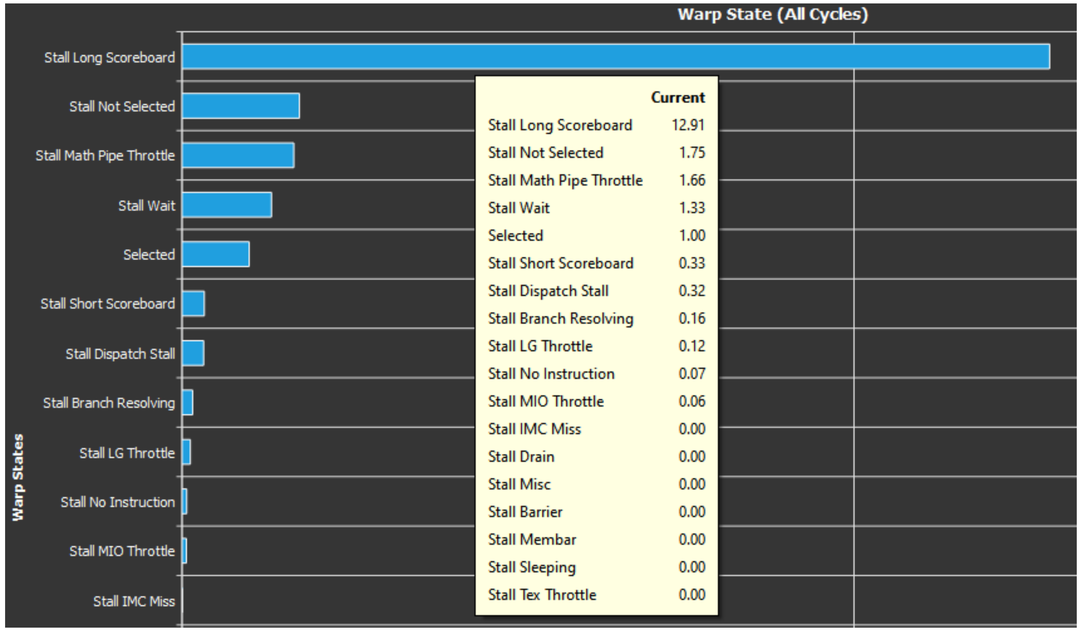
AGENDA

- Using Nsight Compute profiler
- Intro to Profile Devito generated Code with Nsight Compute
- Exercise 1: Coalesced Memory
- Exercise 2: Block Size
- Exercise 3: Reduce Memory Reads
- Exercise 4 : Maximize cache hits
- Exercise 5: Prefetch with Shared Memory

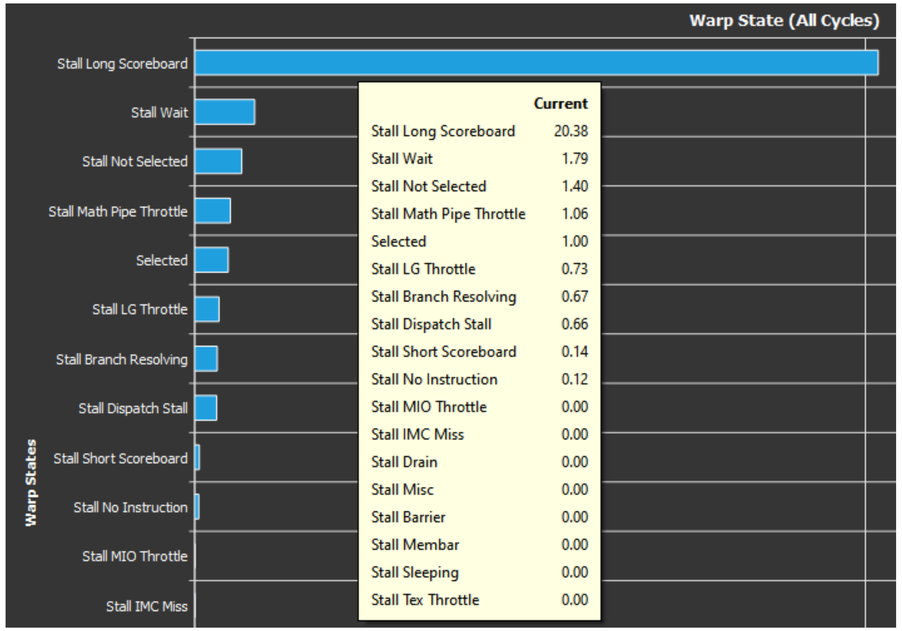
EXERCISE 5 – PREFETCH WITH SHARED MEMORY

What is the stall reason?

- Part 0
  - Long Scoreboard : SM is waiting for memory loads



- Part 1
  - Long scoreboard



EXERCISE 5 - PREFETCH WITH SHARED MEMORY

Prefetch using pipeline

```
#pragma region preload
auto preload = [&](float* smemptr, int index)
{
    if (((index - threadIdx.x) & 3) == 0) // can we do 128 bits loads ?
    {
        if ((threadIdx.x & 3) == 0)
        {
            __pipeline_memcpy_async((float4*)(smemptr + buffsize * (int)smem_fields_0::theta), (float4*)&(p.thetaL0[index]), sizeof(float4));
            __pipeline_memcpy_async((float4*)(smemptr + buffsize * (int)smem_fields_0::phi  ), (float4*)&(p.phiL0[index]), sizeof(float4));
            __pipeline_memcpy_async((float4*)(smemptr + buffsize * (int)smem_fields_0::b    ), (float4*)&(p.bL0[index]), sizeof(float4));
            __pipeline_memcpy_async((float4*)(smemptr + buffsize * (int)smem_fields_0::eps  ), (float4*)&(p.epsL0[index]), sizeof(float4));
            __pipeline_memcpy_async((float4*)(smemptr + buffsize * (int)smem_fields_0::eta  ), (float4*)&(p.etaL0[index]), sizeof(float4));
            __pipeline_memcpy_async((float4*)(smemptr + buffsize * (int)smem_fields_0::f    ), (float4*)&(p.fL0[index]), sizeof(float4));
        }
    } else {
        __pipeline_memcpy_async(smemptr + buffsize * (int)smem_fields_0::theta, &(p.thetaL0[index]), sizeof(float));
        __pipeline_memcpy_async(smemptr + buffsize * (int)smem_fields_0::phi  , &(p.phiL0[index]), sizeof(float));
        __pipeline_memcpy_async(smemptr + buffsize * (int)smem_fields_0::b    , &(p.bL0[index]), sizeof(float));
        __pipeline_memcpy_async(smemptr + buffsize * (int)smem_fields_0::eps  , &(p.epsL0[index]), sizeof(float));
        __pipeline_memcpy_async(smemptr + buffsize * (int)smem_fields_0::eta  , &(p.etaL0[index]), sizeof(float));
        __pipeline_memcpy_async(smemptr + buffsize * (int)smem_fields_0::f    , &(p.fL0[index]), sizeof(float));
    }
    __pipeline_commit();
} ;
#pragma endregion
```

efetch the 6 buffers

uda_pipeline_primitives.h>

ared memory:

- 1 entry per field - coalesced
- 2 buffers: one for current, one for next
- Prefetch one buffer in advance
- (could try more...)

efetching of data:

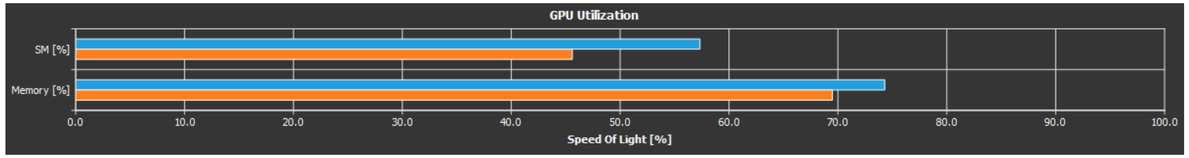
- If pointer aligns to 16 bytes: generate 128bits loads (skips L1 cache)

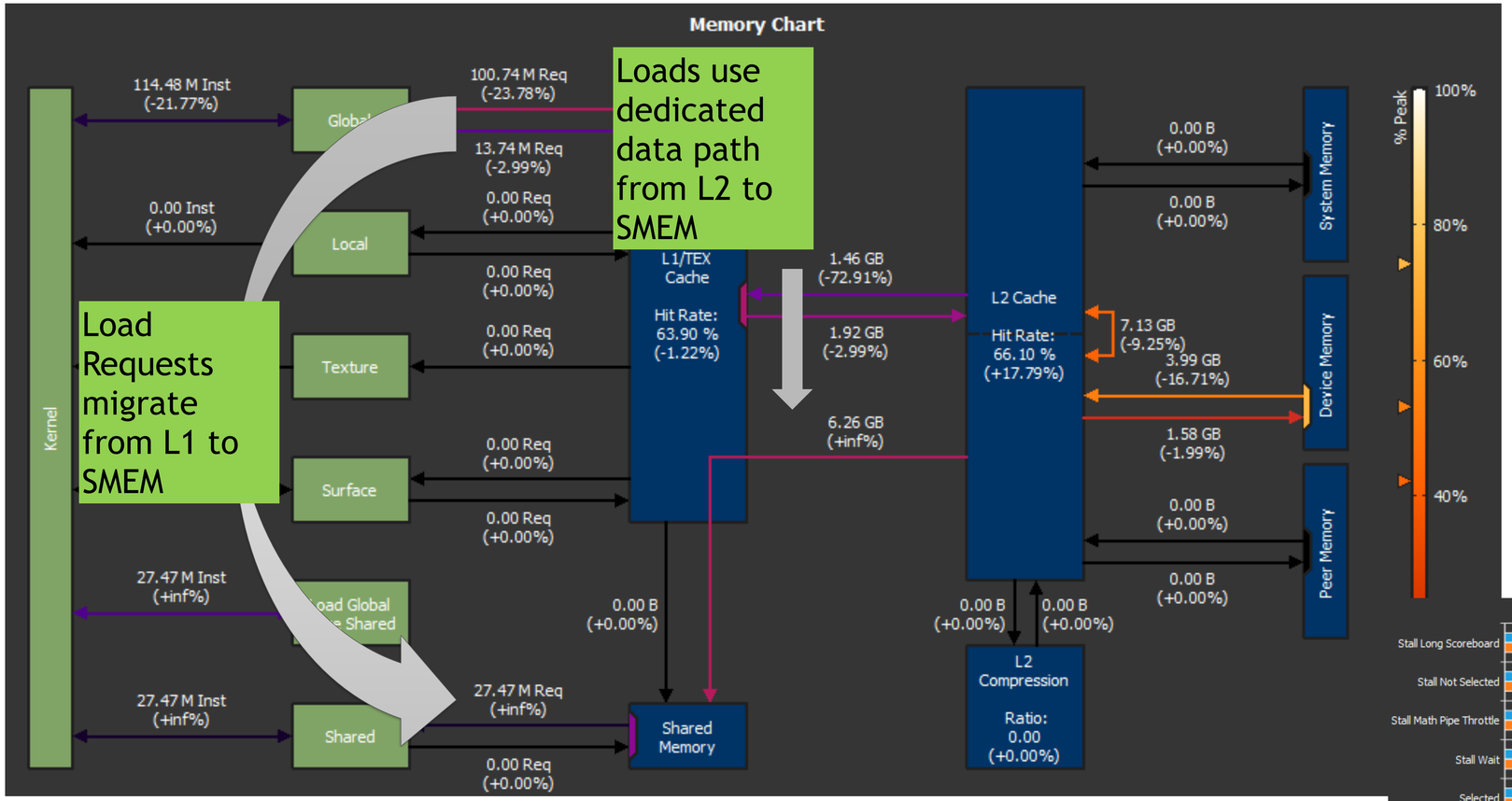
```
if (x < min(to.x, cx+Xchunk-1))
{
    index = p.thetaL0.index(x + 9, y + 8, z + 8);
    preload(&smem[((x+1)&1) * buffsize * (int)smem_fields_0::COUNT + prefetchid], index);
    __pipeline_wait_prior(1);
} else {
    __pipeline_wait_prior(0);
}
```

Before entering the loop, prefetch the first entry,

In the `x` iteration, prefetch the next entry and `wait_prior(1)`

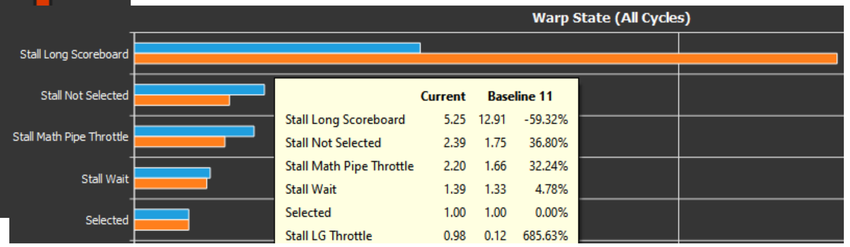
EXERCISE 5 – PREFETCH WITH SHARED MEMORY



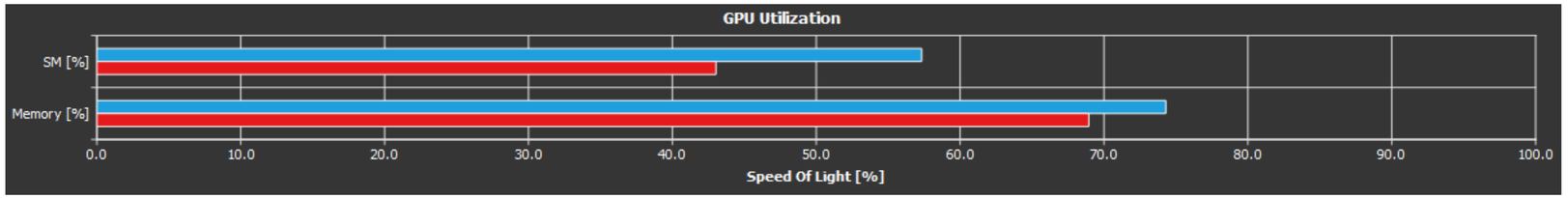


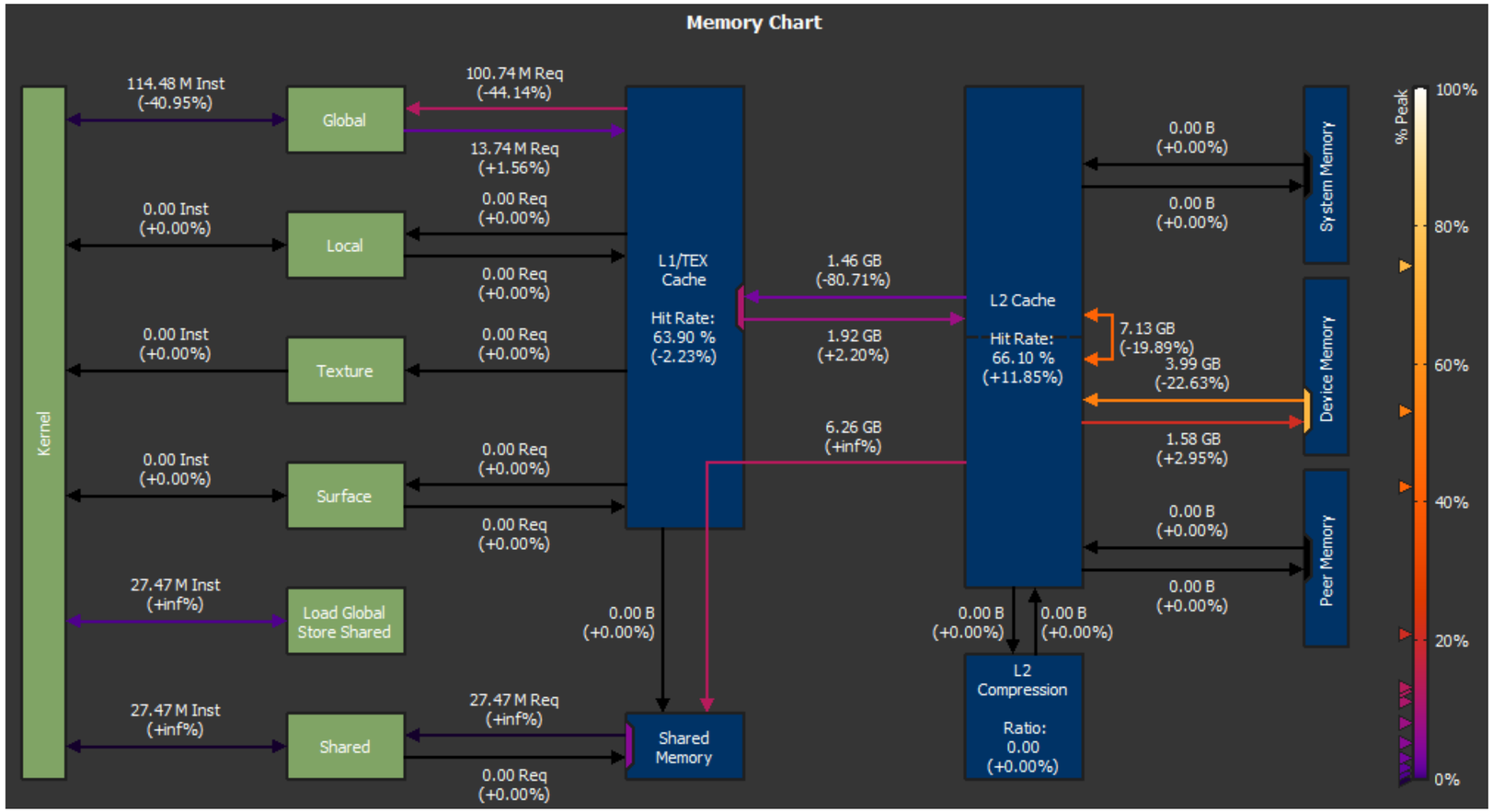
3.94 GB read / 1.68 GB write

- Part 0 – 6.27ms->5.15 ms
- Load request change data path
  - From L2->L1->Kernel
  - To L2->SMEM->Kernel
- Thanks to pipelining latency is reduced (-60% Long Scoreboard)
- Overall reads got improved (getting closer to ideal)

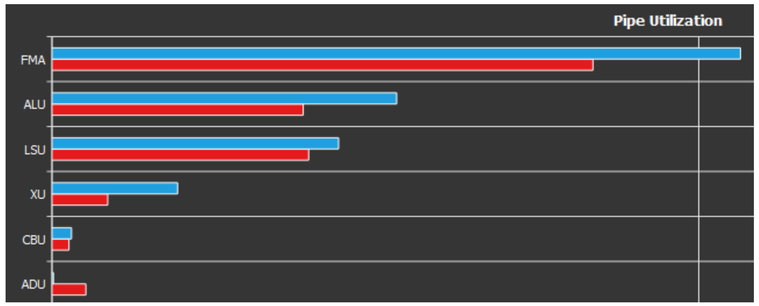


PERFORMANCE MILESTONE

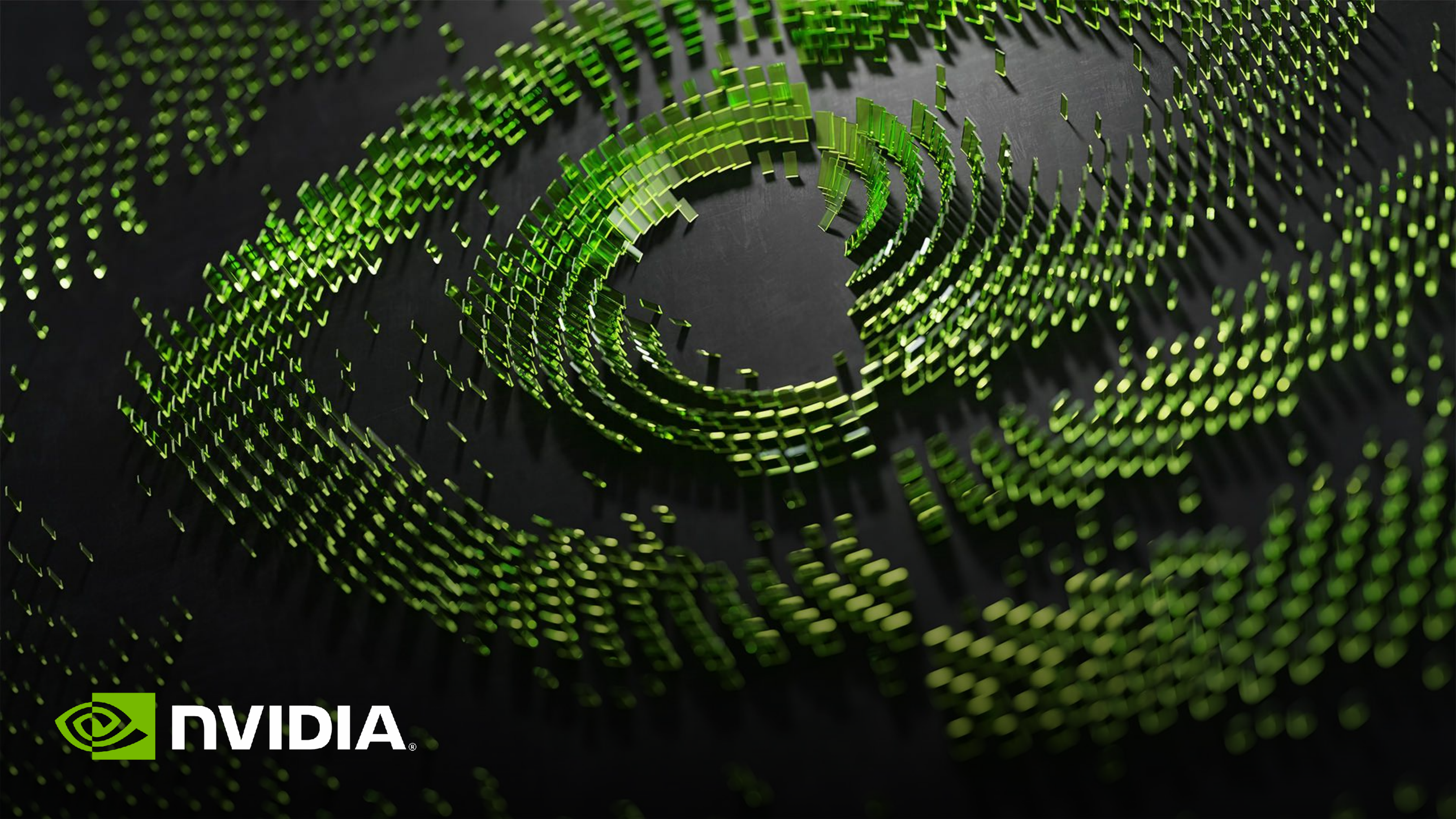
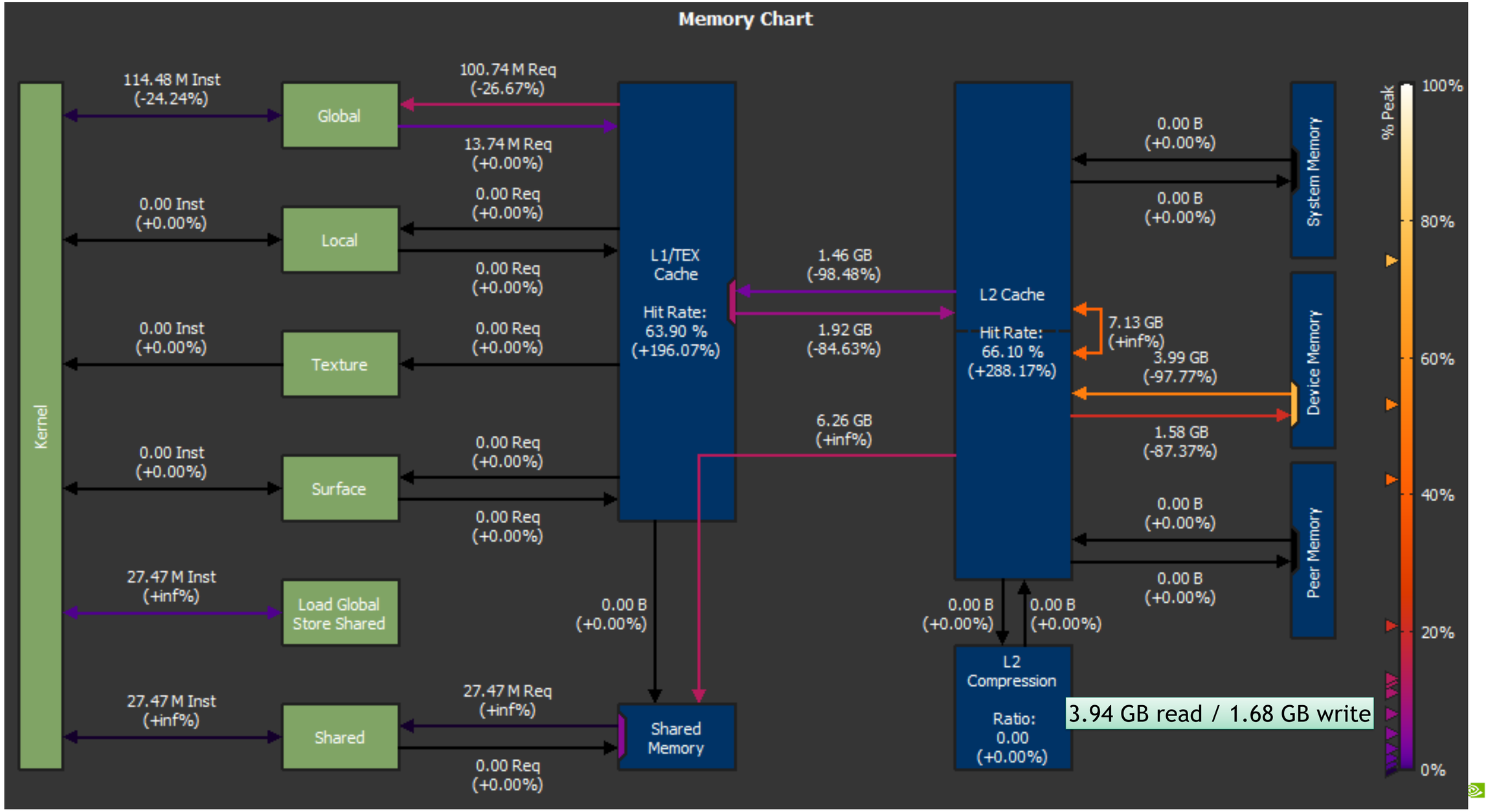




- 6.59ms -> 5.15 ms: -22% execution time
- Used XU function unit better



- Reduced 22.6% global reads (bottleneck here)
- Still Stalls on Long Scoreboard
- Next Steps:
  - Prefetch pressure array as well !


NVIDIA

Current	166 - kernel_section_1_part_0 (17, 17, 17)x(32, 3...	Time: 5.15 msecond	Cycles: 7,297,867	Regs: 61	GPU: Graphics Device	SM Frequency: 1.42 cycle/nsecond	CC: 8.0
Baseline 13	998 - kernel1 (17, 130, 519)x(32, 4, 1)	Time: 856.30 msecond	Cycles: 1,034,552,333	Regs: 40	GPU: NVIDIA TITAN V	SM Frequency: 1.21 cycle/nsecond	CC: 7.0