



# Nsight Compute

## 41TH VI-HPS TUNING WORKSHOP

Felix Schmitt

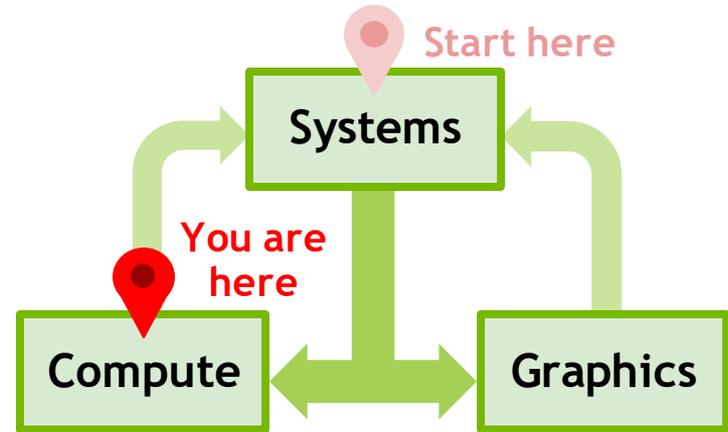
# Nsight Product Family

Nsight Systems - Analyze application algorithms system-wide

Nsight Compute - Analyze CUDA kernels

Nsight Graphics - Debug/analyze graphics workloads

## Workflow





# Overview

# Overview

The screenshot displays the NVIDIA Nsight Compute interface. The main window shows a profile for a kernel named '4969 > KERNEL\_N\_4[SR] O.D->[SR]'. The 'Details' tab is active, showing a table of function calls and their parameters. Below this, several performance analysis sections are visible:

- GPU Speed of Light Throughput:** Shows metrics like Compute (SM) Throughput [N], Memory Throughput [N], L1/TEX Cache Throughput [N], L2 Cache Throughput [N], SM Frequency [cycles/second], and DRAM Throughput [N].
- High Compute Throughput:** A warning indicating that compute is more heavily utilized than memory.
- FP64/32 Utilization:** A warning indicating that the kernel is not fully utilizing the device's peak performance.
- Compute Workload Analysis:** Shows metrics like Executed (pc) Elapsed [inst/cycle], Executed (pc) Active [inst/cycle], and Issue Slots Busy [N].
- Very High Utilization:** A warning indicating that the FP64 is the highest-utilized pipeline.
- Memory Workload Analysis:** Shows metrics like Memory Throughput [bytes/second], L1/TEX Hit Rate [N], and L2 Hit Rate [N].
- Scheduler Statistics:** Shows metrics like Active Warps Per Scheduler [warp], Eligible Warps Per Scheduler [warp], and Issue Warps Per Scheduler [warp].
- Issue Slot Utilization:** A warning indicating that the kernel is not fully utilizing the device's peak performance.

At the bottom, a terminal window shows the command: `GPUblackScholesCallPut(int, float*, float*, float*, float*), 2019-Aug-12 14:44:50, Context 1, Stream 7`. Below the command, a table of performance metrics is displayed:

Metric	Value
Memory Frequency	6.00 Ghz
SOL FB	71.55 %
Elapsed Cycles	749,683 cycle
SM Frequency	1.29 Ghz
Memory [N]	71.55 %
Duration	580.93 usecond
SOL L2	24.53 %
SM Active Cycles	749,656 cycle
SM [N]	69.17 %
SOL TEX	20.49 %

Below the table, a message states: `OK Compute and Memory are well-balanced: To reduce runtime, both computation and memory traffic must be reduced. Check both the "Compute Workload Analysis" and "Memory Workload Analysis" report sections.`

Interactive CUDA Kernel profiler

Targeted metric sections for various performance aspects

Customizable data collection and presentation (tables, charts, ...)

UI and Command Line

Python-based API for guided analysis and post-processing

Support for remote profiling across machines and platforms

# Profiling Activities

## Interactive Profile

## (Non-interactive) Profile

## Command Line

API Stream

15341 \* device\_tea\_leaf\_ppcg\_solve\_update\_r

Next Trigger: nvrtc\_tea\_leaf\_ppcg\_solve\_calc(update)

ID	API Name	Details	Func Return	Func Parameter
43833	cudaMalloc		cudaSuccess(0)	(0x555d75b64838f0)
43834	cuMemAlloc_v2		CUDA_SUCCESS(0)	(0x555d75b64838f0)
43835	cudaMalloc		cudaSuccess(0)	(0x555d75b64838f0)
43836	cuMemAlloc_v2		CUDA_SUCCESS(0)	(0x555d75b64838f0)
43837	cudaMemcpy		cudaSuccess(0)	(0x7feb89f6cc00, 0x)
43838	cuMemcpyHtoD_v2		CUDA_SUCCESS(0)	(0x7feb89f6cc00, 0x)
43839	cudaMemcpy		cudaSuccess(0)	(0x7feb89f6cc00, 0x)
43840	cuMemcpyHtoD_v2		CUDA_SUCCESS(0)	(0x7feb89f6cc00, 0x)
43841	cudaLaunchKernel		cudaSuccess(0)	(0x555d76b0f080, {
43842	cuLaunchKernel		CUDA_SUCCESS(0)	(0x555d76b0f970, 1;
43843	device_tea_leaf_ppcg_solve_init	device_tea_leaf_ppcg_solve_init	cudaSuccess(0)	
43844	cudaLaunchKernel		cudaSuccess(0)	(0x555d75afca40, {
43845	cuLaunchKernel		CUDA_SUCCESS(0)	(0x555d76ec82a0, 5
43846	reduction	reduction	cudaSuccess(0)	
43847	cudaLaunchKernel		cudaSuccess(0)	(0x555d75afca40, {
43848	cuLaunchKernel		CUDA_SUCCESS(0)	(0x555d76ec82a0, 6
43849	reduction	reduction	cudaSuccess(0)	
43850	cudaLaunchKernel		cudaSuccess(0)	(0x555d75afca40, {
43851	cuLaunchKernel		CUDA_SUCCESS(0)	(0x555d76ec82a0, 1
43852	reduction	reduction	cudaSuccess(0)	
43853	cudaMemcpy		cudaSuccess(0)	(0x7f02a710a08, 0x)
43854	cuMemcpyHtoH_v2		CUDA_SUCCESS(0)	(0x7f02a710a08, 0x)
43855	cudaLaunchKernel		cudaSuccess(0)	(0x555d76b0f040, 1
43856	cuLaunchKernel		CUDA_SUCCESS(0)	(0x555d76b0b610, 1
43857	device_tea_leaf_ppcg_solve_init_new	device_tea_leaf_ppcg_solve_init_sd_new	cudaSuccess(0)	
43858	cudaLaunchKernel		cudaSuccess(0)	(0x555d76b0f020, {
43859	cuLaunchKernel		cudaSuccess(0)	(0x555d76f0eb50, 1

Resources

CUDA: Memory Allocations

Enter filter, e.g. 3(ID) >= 10

ID	API Call ID	Allocation type	Address	Size Requested	Context	Device ID
Total allocations: 41 Total size: 2.63 Gbytes						
1	683	UNIFIED MEMORY ALLOC	0x7fec2000000	32 bytes	0x555d76933600	0
2	655	DEVICE MEMORY ALLOC	0x7fec2400000	122.19 Mbytes	0x555d76933600	0
3	666	DEVICE MEMORY ALLOC	0x7fec2200000	122.19 Mbytes	0x555d76933600	0
4	677	DEVICE MEMORY ALLOC	0x7fec1400000	122.19 Mbytes	0x555d76933600	0
5	688	DEVICE MEMORY ALLOC	0x7fec1200000	122.19 Mbytes	0x555d76933600	0

Activity

Interactive Profile

Profile an application using the command line profiler. All GPU workloads are serialized. Note: Attach is not supported for this activity.

Supported APIs: CUDA

Common Filter Sections Sampling Other

Output File: report\_%.xml

Force Override: Yes

Target Processes: Application Only

Replay Mode: Kernel

Application Replay Match: Grid

Application Replay Buffer: File

Command Line: /tmp/var/target/linux-desktop-glibc\_2\_1\_1\_3-x64/ncu - -export /tmp/var/report\_%.xml --force-override --target-processes application-only --replay-mode kernel --kernel-name-base function --launch-skip-before-match 0 --launch-

Cancel Reset Activity Launch

```
~/working/git/TeaLeaf_CUDA5 /tmp/var/target/linux-desktop-glibc_2_1_1_3-x64/ncu -c 2 -k
"regex:device_tea_leaf_ppcg_solve_calc(update)." -- //tea_leaf
==PROF== Connected to process 15827 (/home/fschmitt/working/git/TeaLeaf_CUDA/tea_leaf)
--
  Output file tea.out opened. All output will go there.
  CUDA in rank 0 using NVIDIA GeForce RTX 2080 Ti
  Solver to use: PPGC
  Preconditioner to use: None
  Step 1 time 0.000000 timestep 4.00E-03
  Switching after 990 CG its, error 0.991110E+00
  Eigen min 0.106142E+01 Eigen max 0.537593E+05 Condition number 50648.403204 Error 0.995546E+00
  ==PROF== Profiling "device_tea_leaf_ppcg_solve_update_r": 0%...50%...100% - 8 passes
  ==PROF== Profiling "device_tea_leaf_ppcg_solve_calc_sd_new": 0%...50%...100% - 8 passes
--
[15827] tea_leaf@127.0.0.1
device_tea_leaf_ppcg_solve_update_r(kernel_info_t, double *, const double *, const double *, const double *), 2021-Dec-
14 14:02:35, Context 1, Stream 7
  Section: GPU Speed Of Light Throughput
-----
DRAM Frequency                                cycle/msecond                                6.63
SM Frequency                                  cycle/msecond                                1.33
Elapsed Cycles                                cycle                                        1,457,762
Memory [%]                                    %                                             91.78
DRAM Throughput                               %                                             91.78
Duration                                       msecond                                       1.09
L1/TEX Cache Throughput                       %                                             28.58
L2 Cache Throughput                           %                                             39.02
SM Active Cycles                              cycle                                        1,451,446.41
Compute (SM) [%]                              %                                             86.51
-----
INF The kernel is utilizing greater than 80.0% of the available compute or memory performance of the device. To
further improve performance, work will likely need to be shifted from the most utilized to another unit.
Start by analyzing workloads in the Memory Workload Analysis section.
```

# Profiler Report

Page: Summary Launch: 0 - 43843 - device\_tea\_leaf\_ppcg\_sol Add Baseline Apply Rules Occupancy Calculator Copy as Image

Launch	Time	Cycles	Regs	GPU	SM Frequency	CC	Process
<b>Current</b> 43843 - device_tea_leaf_ppcg_solve_init (126, 1001, 1)x(32, 4, 1)	217.63 usecond	297,114	40	0 - NVIDIA GeForce RTX 2080 Ti	1.36 cycle/nsecond	7.5	[15958] tea_Leaf

↶

ID	Time	API Call ID	Function Name	Demangled N; Process	Device Name	Grid Size	Block Size	Cycles [cycle]	Duration [msecond]	Compute Throughput [%]	Memc
0	2021-Dec-1...	43843	device_tea_leaf_ppcg...	device_te... [15958] tea_leaf	NVIDIA GeForce...	126, 1001, 1		32, 4, 1	297,114	0.22	77.89
1	2021-Dec-1...	43857	device_tea_leaf_ppcg_sol...	device_tea_... [15958] tea_leaf	NVIDIA GeForce RT...	126, 1001, 1		32, 4, 1	1,264,921	0.94	54.78
2	2021-Dec-1...	43860	device_tea_leaf_ppcg_sol...	device_tea_... [15958] tea_leaf	NVIDIA GeForce RT...	126, 1001, 1		32, 4, 1	1,462,446	1.07	86.22
3	2021-Dec-1...	43863	device_tea_leaf_ppcg_sol...	device_tea_... [15958] tea_leaf	NVIDIA GeForce RT...	126, 1001, 1		32, 4, 1	1,443,836	1.06	23.81

Page: Details Launch: 0 - 43843 - device\_tea\_leaf\_ppcg\_sol Add Baseline Apply Rules Occupancy Calculator Copy as Image

Launch	Time	Cycles	Regs	GPU	SM Frequency	CC	Process
<b>Current</b> 43843 - device_tea_leaf_ppcg_solve_init (126, 1001, 1)x(32, 4, 1)	217.63 usecond	297,114	40	0 - NVIDIA GeForce RTX 2080 Ti	1.36 cycle/nsecond	7.5	[15958] tea_Leaf

⊕ ⊖ 🔍 ⓘ

### GPU Speed Of Light Throughput

Metric	Value	Unit
Compute (SM) Throughput [%]	77.89	Duration [usecond]
Memory Throughput [%]	45.03	Elapsed Cycles [cycle]
L1/TEX Cache Throughput [%]	68.22	SM Active Cycles [cycle]
L2 Cache Throughput [%]	2.30	SM Frequency [cycle/nsecond]
DRAM Throughput [%]	0.12	DRAM Frequency [cycle/nsecond]

**High Compute Throughput** Compute is more heavily utilized than Memory: Look at the [Compute Workload Analysis](#) report section to see what the compute pipelines are spending their time doing. Also, consider whether any computation is redundant and could be reduced or moved to look-up tables.

**FP64/32 Utilization** The ratio of peak float (fp32) to double (fp64) performance on this device is 32:1. The kernel achieved 0% of this device's fp32 peak performance and 19% of its fp64 peak performance. If [Compute Workload Analysis](#) determines that this kernel is fp64 bound, consider using 32-bit precision floating point operations to improve its performance. See the [Kernel Profiling Guide](#) for mode details on roffline analysis.

### Compute Workload Analysis

Executed Ipc Elapsed [inst/cycle]	1.25	SM Busy [%]	78.61
Executed Ipc Active [inst/cycle]	1.27	Issue Slots Busy [%]	31.65
Issued Ipc Active [inst/cycle]	1.27		

# Profiler Report

Selected result

Metric values

Page: Details Launch: 0 - 43843 - device\_tea\_leaf\_ppcg\_sol Add Baseline Apply Rules Occupancy Calculator Copy as Image

Launch	Time	Cycles	Regs	GPU	SM Frequency	CC	Process
Current	43843 - device_tea_leaf_ppcg_solve_init (126, 1001, 1)x(32, 4, 1)	217.63 usecond	297,114	40	0 - NVIDIA GeForce RTX 2080 Ti	1.36 cycle/nsecond	7.5 [15958] tea_leaf

GPU Speed Of Light Throughput

Metric	Value	Metric	Value
Compute (SM) Throughput [%]	77.89	Duration [usecond]	217.63
Memory Throughput [%]	45.03	Elapsed Cycles [cycle]	297,114
L1/TEX Cache Throughput [%]	68.22	SM Active Cycles [cycle]	293,385.19
L2 Cache Throughput [%]	2.30	SM Frequency [cycle/nsecond]	1.36
DRAM Throughput [%]	0.12	DRAM Frequency [cycle/nsecond]	6.79

**High Compute Throughput** Compute is more heavily utilized than Memory: Look at the [Compute Workload Analysis](#) report section to see what the compute pipelines are spending their time doing. Also, consider whether any computation is redundant and could be reduced or moved to look-up tables.

**FP64/32 Utilization** The ratio of peak float (fp32) to double (fp64) performance on this device is 32:1. The kernel achieved 0% of this device's fp32 peak performance and 19% of its fp64 peak performance. If [Compute Workload Analysis](#) determines that this kernel is fp64 bound, consider using 32-bit precision floating point operations to improve its performance. See the [Kernel Profiling Guide](#) for mode details on roofline analysis.

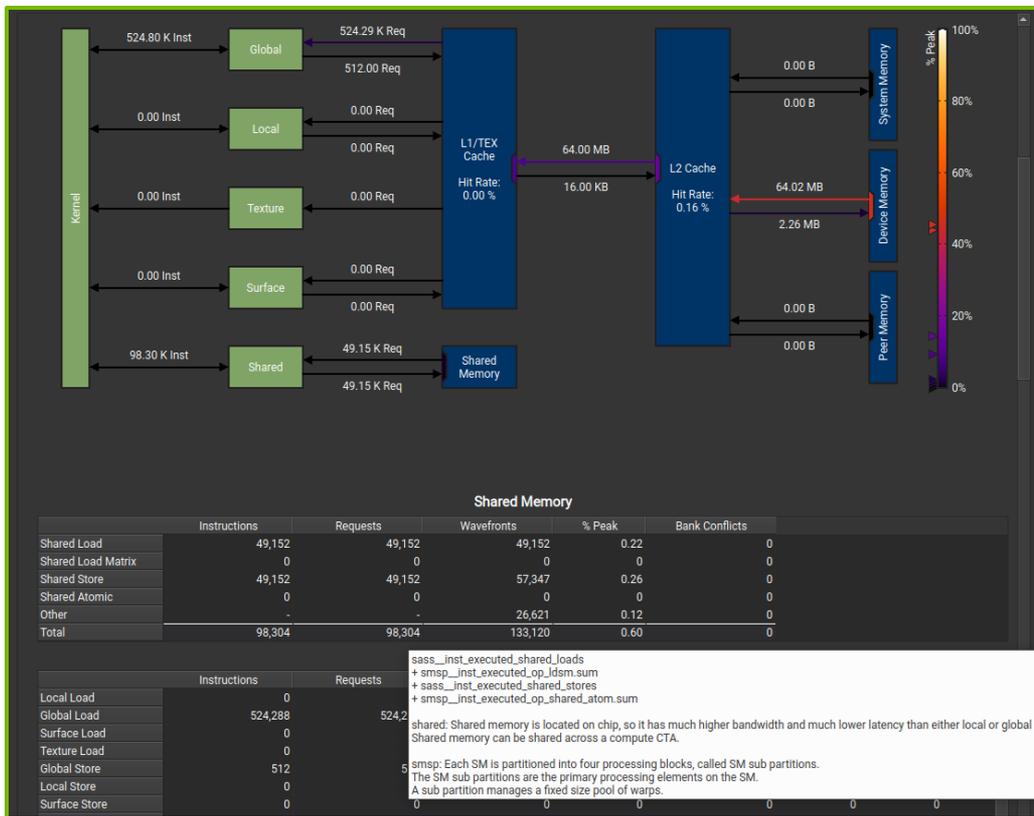
**Compute Workload Analysis**

Metric	Value	Metric	Value
Executed ipc Elapsed [inst/cycle]	1.25	SM Busy [%]	78.61
Executed ipc Active [inst/cycle]	1.27	Issue Slots Busy [%]	31.65
Issued ipc Active [inst/cycle]	1.27		

Expandable Sections

Expert Analysis (Rules)

# Profiler Report

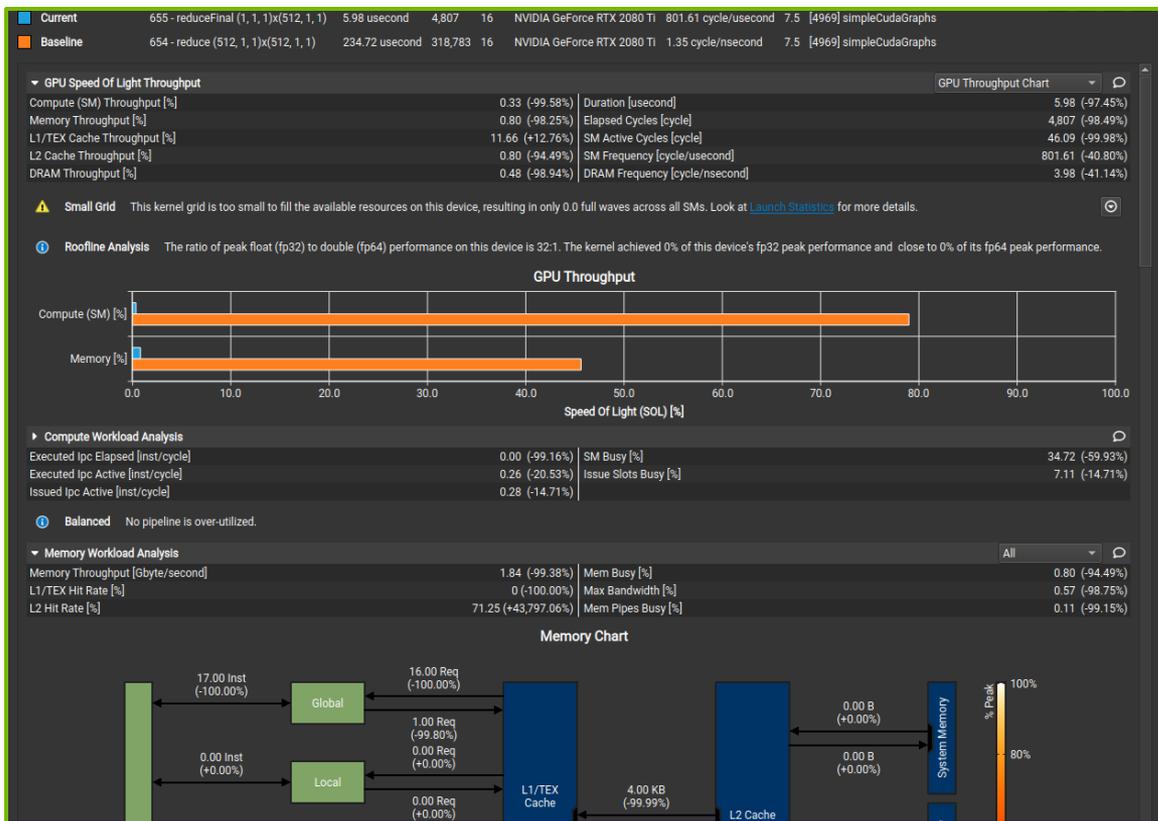


Detailed memory workload analysis chart and tables

Shows transferred data or throughputs

Tooltips provide metric names, calculation formulas and detailed background info

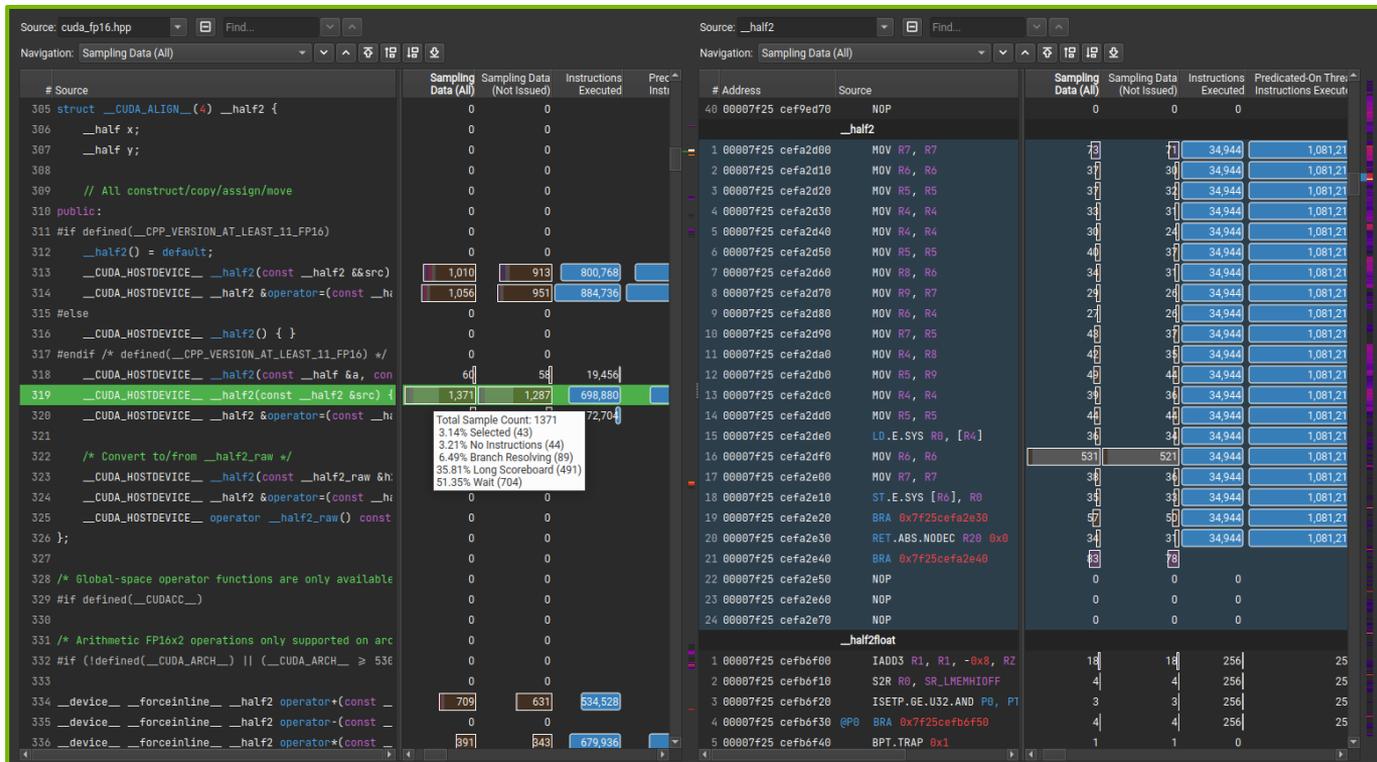
# Profiler Report



Comparison of results directly within the tool with "Baselines"

Supported across kernels, reports, and GPU architectures

# Profiler Report



Source/PTX/SASS analysis and correlation

Source metrics per instruction and aggregated (e.g. PC sampling data)

Metric heatmap

# Occupancy Calculator

Compute Capability:

Shared Memory Size Config (bytes):

CUDA version:

Global Load Cache Mode:

Threads Per Block:

Registers Per Thread:

Shared Memory Per Block (bytes):

Tables

Graphs

GPU Data

**Occupancy Data:**

Property	Value
Active Threads per Multiprocessor	1024
Active Warps per Multiprocessor	32
Active Thread Blocks per Multiprocessor	8
Occupancy of each Multiprocessor	100 %

**Allocated Resources:**

Resources	Per Block	Limit Per SM	Allocatable Blocks Per SM
Warps (Threads Per Block / Threads Per Warp)	4	32	8
Registers (Warp limit per SM due to per-warp reg count)	4	48	12
Shared Memory (Bytes)	3072	65536	21

**Occupancy Limiters:**

Limited By	Blocks per SM	Warps Per Block	Warps Per SM
Max Warps or Max Blocks per Multiprocessor	8	4	32
Registers per Multiprocessor	12		
Shared Memory per Multiprocessor	21		

**Physical Limit of GPU (7.5):**

Property	Limit
Threads per Warp	32
Max Warps per Multiprocessor	32
Max Thread Blocks per Multiprocessor	16
Max Threads per Multiprocessor	1024
Maximum Thread Block Size	1024
Registers per Multiprocessor	65536
Max Registers per Thread Block	65536
Max Registers per Thread	255
Shared Memory per Multiprocessor (bytes)	65536
Max Shared Memory per Block	65536
Register Allocation Unit Size	256
Register Allocation Granularity	warp
Shared Memory Allocation Unit Size	256
Warp Allocation Granularity	4
Shared Memory Per Block (bytes) (CUDA runtime use)	0

The occupancy is limited by block size

11



# Hands-On with TeaLeaf\_Cuda

# Get the application

## Get the source code

```
$ git clone --depth 1 https://github.com/UK-MAC/TeaLeaf_CUDA
```

## Get compiler, MPI and CUDA using modules

```
module load Stages/2022
module load NVHPC/22.1
module load OpenMPI/4.1.2
module load Nsight-Compute/2022.1.0
```

Update Makefile for the target architecture (e.g. AMPERE, SM 8.0) and compiler/libraries, as necessary

## Build the application

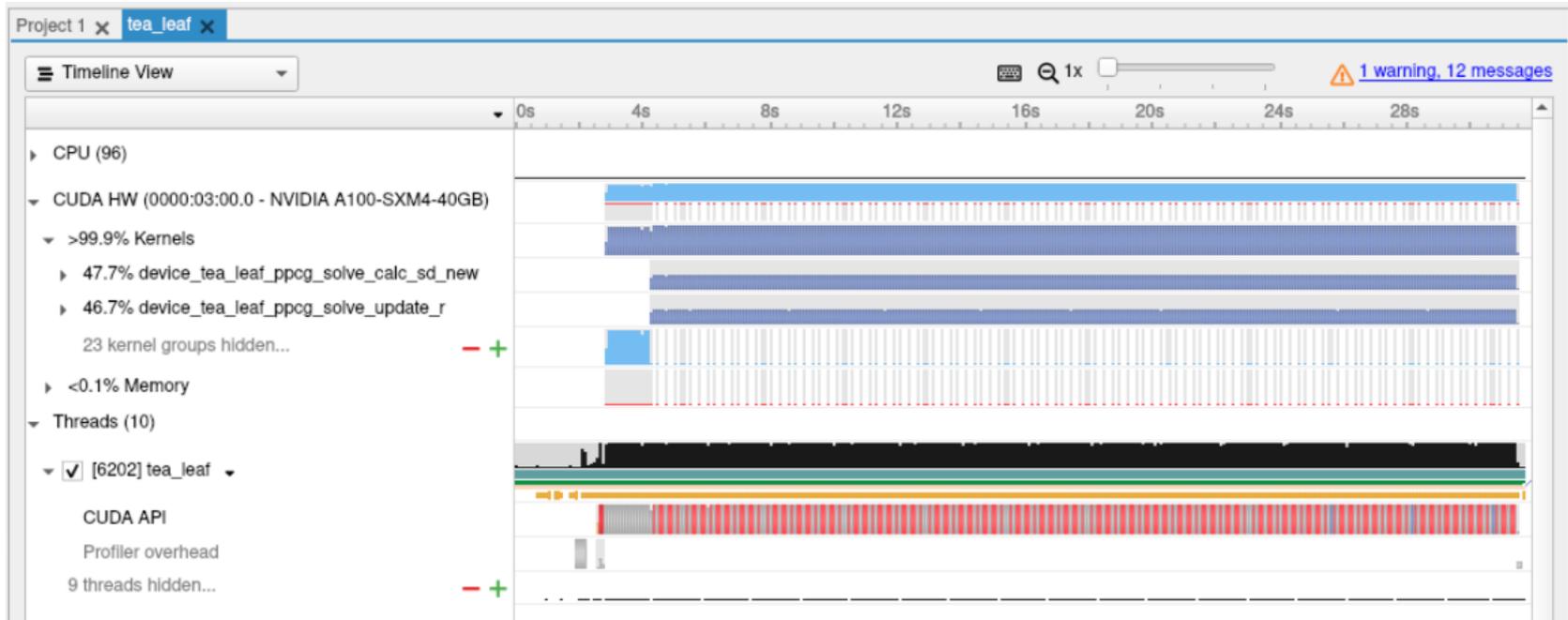
```
$ make
```

Better, use the pre-compiled script and binary from /p/project/training2123/work/schmitt5

```
$ source load_modules.sh
$ TeaLeaf_CUDA/tea_leaf
```

# Overview with Nsight Systems

Profile with Nsight Systems to identify best CUDA kernel optimization targets  
Focus on `device_tea_leaf_ppcg_solve_(calc|update).*` kernels



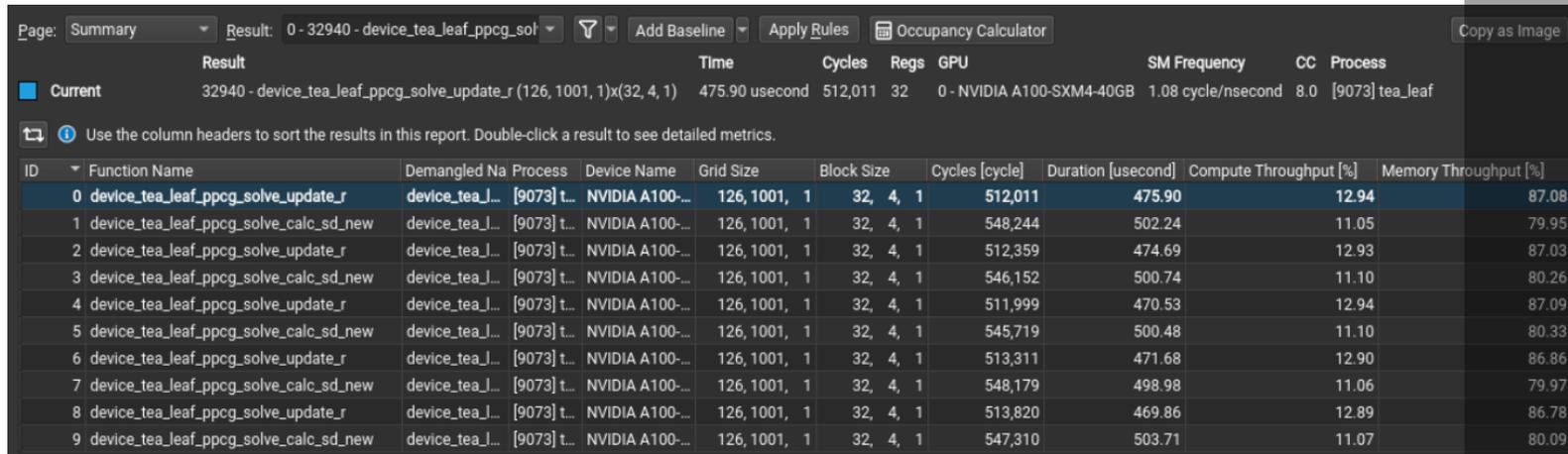
# Profiling with Nsight Compute

Collect 10 instances of those kernels with ncu, full set of metrics  
Inspect the resulting report in the Nsight Compute UI (ncu-ui)

```
ncu --set full -k "regex:device_tea_leaf_ppcg_solve_(calc|update).*" -c 10 -f -o tea_leaf_%i  
./tea_leaf
```

Summary page confirms that all instances of each respective kernel have similar performance characteristics - focus on a single instance for each

# Live



The screenshot shows the Nsight Compute UI with a summary table of kernel performance metrics. The table has columns for ID, Function Name, Demangled Name, Process, Device Name, Grid Size, Block Size, Cycles [cycle], Duration [usecond], Compute Throughput [%], and Memory Throughput [%]. The first row is highlighted in blue, indicating the current selection.

ID	Function Name	Demangled Name	Process	Device Name	Grid Size	Block Size	Cycles [cycle]	Duration [usecond]	Compute Throughput [%]	Memory Throughput [%]
0	device_tea_leaf_ppcg_solve_update_r	device_tea_l...	[9073] t...	NVIDIA A100-...	126, 1001, 1	32, 4, 1	512,011	475.90	12.94	87.08
1	device_tea_leaf_ppcg_solve_calc_sd_new	device_tea_l...	[9073] t...	NVIDIA A100-...	126, 1001, 1	32, 4, 1	548,244	502.24	11.05	79.95
2	device_tea_leaf_ppcg_solve_update_r	device_tea_l...	[9073] t...	NVIDIA A100-...	126, 1001, 1	32, 4, 1	512,359	474.69	12.93	87.03
3	device_tea_leaf_ppcg_solve_calc_sd_new	device_tea_l...	[9073] t...	NVIDIA A100-...	126, 1001, 1	32, 4, 1	546,152	500.74	11.10	80.26
4	device_tea_leaf_ppcg_solve_update_r	device_tea_l...	[9073] t...	NVIDIA A100-...	126, 1001, 1	32, 4, 1	511,999	470.53	12.94	87.09
5	device_tea_leaf_ppcg_solve_calc_sd_new	device_tea_l...	[9073] t...	NVIDIA A100-...	126, 1001, 1	32, 4, 1	545,719	500.48	11.10	80.33
6	device_tea_leaf_ppcg_solve_update_r	device_tea_l...	[9073] t...	NVIDIA A100-...	126, 1001, 1	32, 4, 1	513,311	471.68	12.90	86.86
7	device_tea_leaf_ppcg_solve_calc_sd_new	device_tea_l...	[9073] t...	NVIDIA A100-...	126, 1001, 1	32, 4, 1	548,179	498.98	11.06	79.97
8	device_tea_leaf_ppcg_solve_update_r	device_tea_l...	[9073] t...	NVIDIA A100-...	126, 1001, 1	32, 4, 1	513,820	469.86	12.89	86.78
9	device_tea_leaf_ppcg_solve_calc_sd_new	device_tea_l...	[9073] t...	NVIDIA A100-...	126, 1001, 1	32, 4, 1	547,310	503.71	11.07	80.09

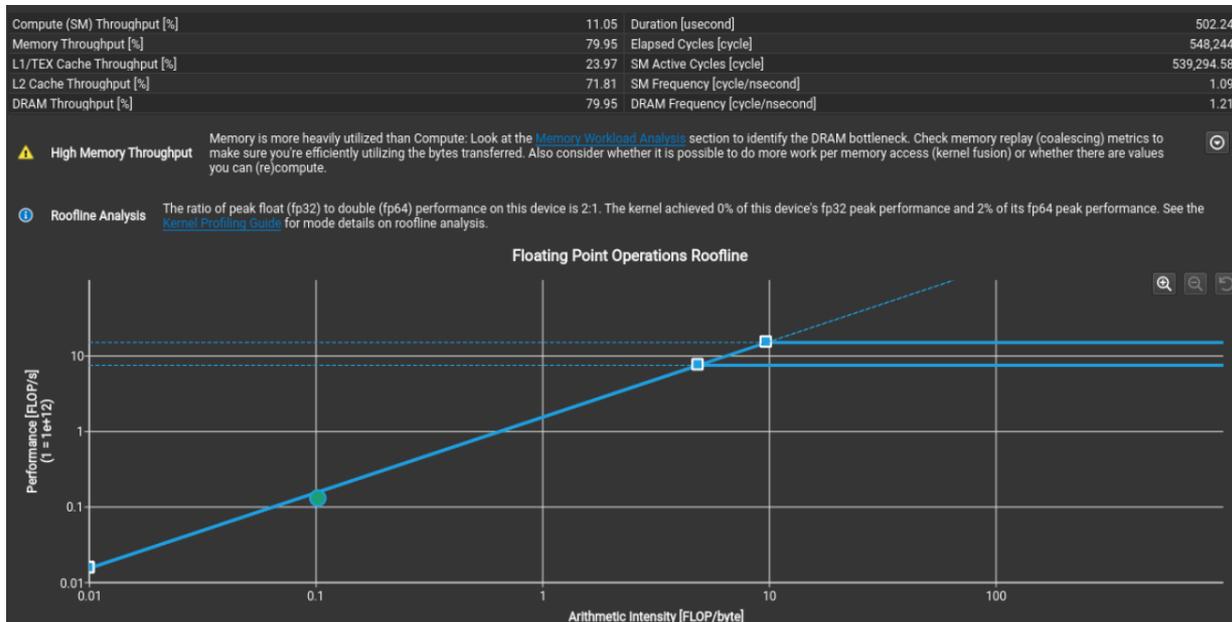
# Analyzing calc kernel

Switch to the *Details* page and select the second kernel (with slightly worse throughputs).

Memory units are over-utilized.

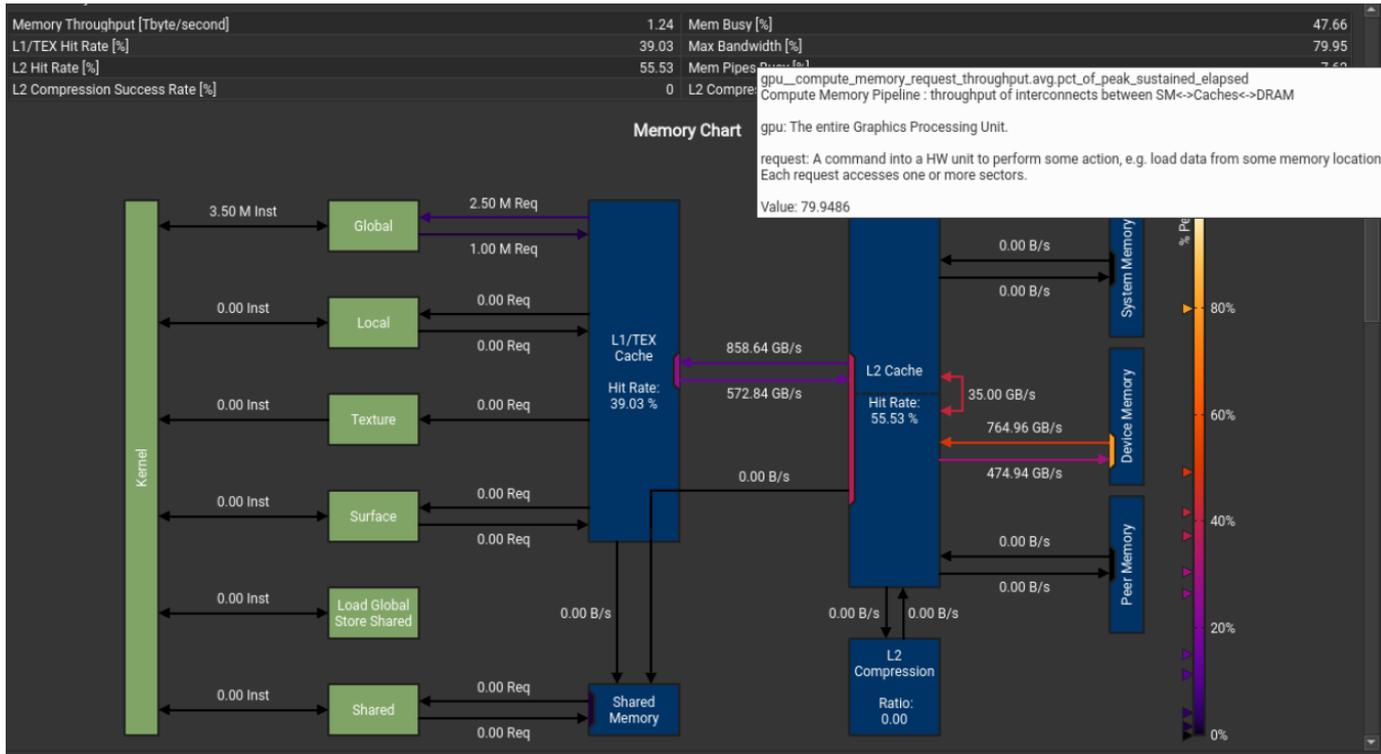
Roofline shows that floating point performance is memory-bound (left of ridge point)

Similar in Compute Workload chart: FP pipelines are less than 5% utilized.



# Analyzing calc kernel

MWA table shows bandwidth 80% utilized, chart shows high Device-to-L2 utilization

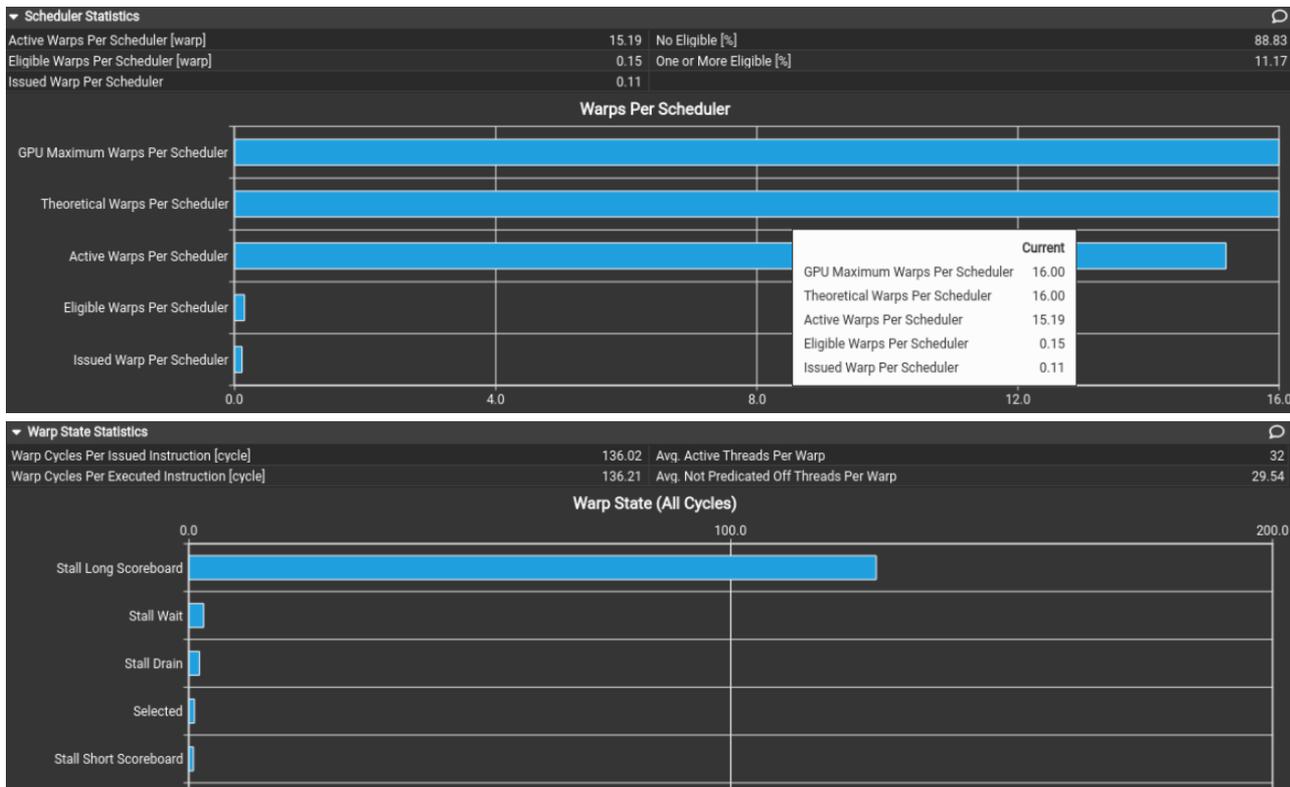


# Analyzing calc kernel

Scheduler stats show low eligible/issued, need to check stall reasons

Good achieved occupancy, doesn't appear to be the issue

Stall reasons dominated by long scoreboard, locate using Source Counters section



# Analyzing calc kernel

MWA found sub-optimal cache access patterns, locate using Source Counters section

Source Counters show uncoalesced accesses and location of the stalls

Jump to Source page via this link

**L1TEX Global Store Access Pattern** The memory access pattern for global stores in L1TEX might not be optimal. On average, this kernel accesses 8.0 bytes per thread per memory request; but the address pattern, possibly caused by the stride between threads, results in 9.0 sectors per request, or  $9.0 \times 32 = 288.0$  bytes of cache data transfers per request. The optimal thread address pattern for 8.0 byte accesses would result in  $8.0 \times 32 = 256.0$  bytes of cache data transfers per request, to maximize L1TEX cache performance. Check the [Source Counters](#) section for uncoalesced global stores.

**L2 Store Access Pattern** The memory access pattern for stores from L1TEX to L2 is not optimal. The granularity of an L1TEX request to L2 is a 128 byte cache line. That is 4 consecutive 32-byte sectors per L2 request. However, this kernel only accesses an average of 3.0 sectors out of the possible 4 sectors per cache line. Check the [Source Counters](#) section for uncoalesced stores and try to minimize how many cache lines need to be accessed per memory request.

**L2 Load Access Pattern** The memory access pattern for loads from L1TEX to L2 is not optimal. The granularity of an L1TEX request to L2 is a 128 byte cache line. That is 4 consecutive 32-byte sectors per L2 request. However, this kernel only accesses an average of 3.0 sectors out of the possible 4 sectors per cache line. Check the [Source Counters](#) section for uncoalesced loads and try to minimize how many cache lines need to be accessed per memory request.

**Source Counters** All

Branch Instructions [inst] 2,518,016 Branch Efficiency [%] 100  
Branch Instructions Ratio [%] 0.10 Avg. Divergent Branches 0

**Uncoalesced Global Accesses** This kernel has uncoalesced global accesses resulting in a total of 2500000 excessive sectors (11% of the total 23500000 sectors). Check the L2 Theoretical Sectors Global Excessive table for the primary source locations. The [CUDA Programming Guide](#) had additional information on reducing uncoalesced device memory accesses.

**L2 Theoretical Sectors Global Excessive**

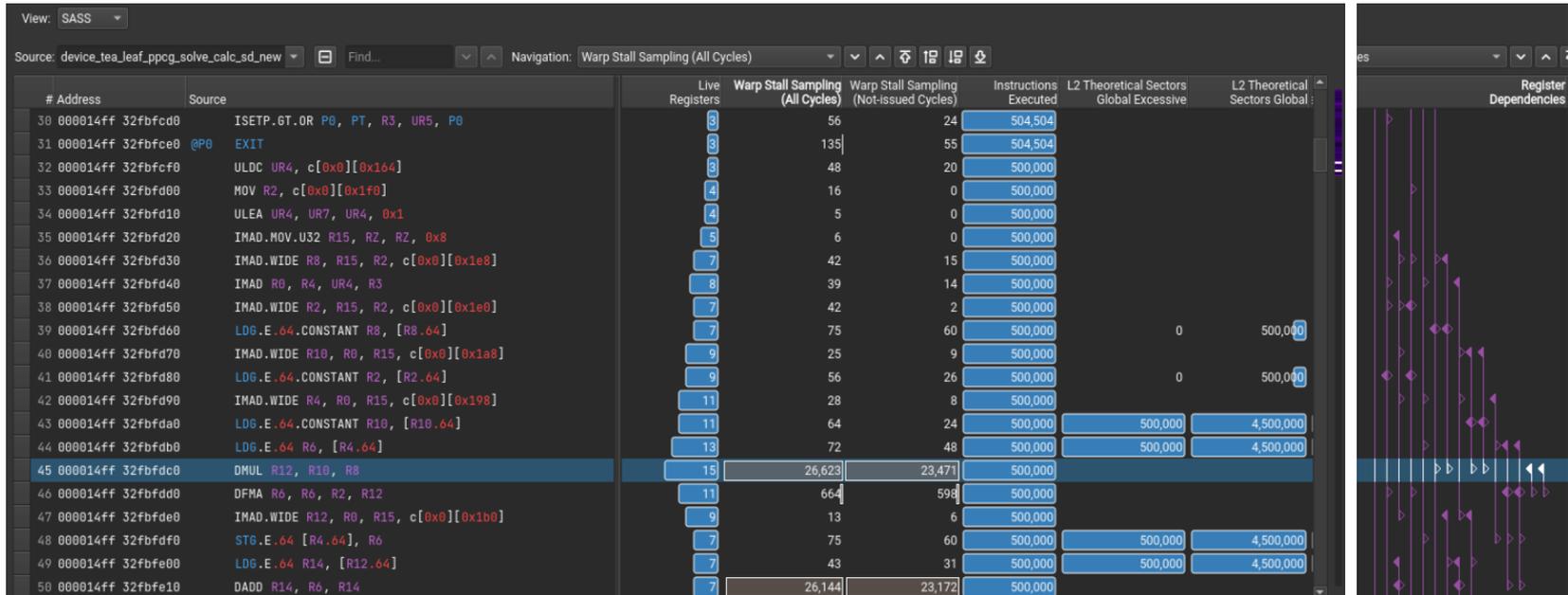
Location	Value	Value (%)
<a href="#">0x14ff32fbfe20 in device_tea_leaf_ppcg_solve_calc_sd_new</a>	500,000	20
<a href="#">0x14ff32fbfe00 in device_tea_leaf_ppcg_solve_calc_sd_new</a>	500,000	20
<a href="#">0x14ff32fbfd0 in device_tea_leaf_ppcg_solve_calc_sd_new</a>	500,000	20
<a href="#">0x14ff32fbfdb0 in device_tea_leaf_ppcg_solve_calc_sd_new</a>	500,000	20
<a href="#">0x14ff32fbfda0 in device_tea_leaf_ppcg_solve_calc_sd_new</a>	500,000	20

**Warp Stall Sampling (All Cycles)**

Location	Value	Value (%)
<a href="#">0x14ff32fbfd0 in device_tea_leaf_ppcg_solve_calc_sd_new</a>	26,623	47
<a href="#">0x14ff32fbfe10 in device_tea_leaf_ppcg_solve_calc_sd_new</a>	26,144	46
<a href="#">0x14ff32fbfe40 in device_tea_leaf_ppcg_solve_calc_sd_new</a>	1,407	3
<a href="#">0x14ff32fbfd0 in device_tea_leaf_ppcg_solve_calc_sd_new</a>	664	1
<a href="#">0x14ff32fbfb80 in device_tea_leaf_ppcg_solve_calc_sd_new</a>	334	1

# Analyzing calc kernel

Stalled at DMUL instruction, waiting for LDG (load global) in line 43 (via register R10)  
LDG instructions are uncoalesced  
Lots of excessive (non-ideal) L2 sector accesses



# Analyzing calc kernel

Where is this in the code  
(no CUDA-C source)  
correlation?

Need to add `-lineinfo` flag  
in Makefile during compilation  
on line 145 (NV\_FLAGS),  
re-compile, re-run

Consider using  
`--import-source` yes

The image shows two IDE windows side-by-side. The left window displays C++ source code for a kernel, with line numbers 231 through 250 visible. The right window displays the corresponding assembly code, with addresses and instructions listed. A 'Warp Stall Sampling' overlay is present on the right side of the assembly window, showing cycle counts for various instructions.

Address	Source	Warp Stall Sampling (All Cycles)
37	00001476 e6fc4f40 IMAD R0, R4, UR4, R3	44
38	00001476 e6fc4f50 IMAD.WIDE R2, R15, R2, c[0x0][0x1e0]	36
39	00001476 e6fc4f60 LDG.E.64.CONSTANT R8, [R8.64]	81
40	00001476 e6fc4f70 IMAD.WIDE R10, R0, R15, c[0x0][0x1a8]	23
41	00001476 e6fc4f80 LDG.E.64.CONSTANT R2, [R2.64]	49
42	00001476 e6fc4f90 IMAD.WIDE R4, R0, R15, c[0x0][0x198]	23
43	00001476 e6fc4fa0 LDG.E.64.CONSTANT R10, [R10.64]	66
44	00001476 e6fc4fb0 LDG.E.64 R6, [R4.64]	79
45	00001476 e6fc4fc0 DMUL R12, R10, R8	26,586
46	00001476 e6fc4fd0 DFMA R0, R6, R2, R12	650
47	00001476 e6fc4fe0 IMAD.WIDE R12, R0, R15, c[0x0][0x1b0]	20
48	00001476 e6fc4ff0 STG.E.64 [R4.64], R6	92
49	00001476 e6fc5000 LDG.E.64 R14, [R12.64]	40
50	00001476 e6fc5010 DADD R14, R6, R14	26,176
51	00001476 e6fc5020 STG.E.64 [R12.64], R14	107
52	00001476 e6fc5030 EXIT	8
53	00001476 e6fc5040 ULDC UR7, c[0x0][0x170]	1,408
54	00001476 e6fc5050 ULDC UR4, c[0x0][0x16c]	0
55	00001476 e6fc5060 UIADD3 UR4, UR7, UR4, URZ	0
56	00001476 e6fc5070 ULDC UR6, c[0x0][0x18c]	0

# Further CLI exercises

Check collected report on command line using `ncu -i` and `--page` for comparison with the UI.

Update `tea_leaf_kernel_cuda.cu` with a `nvtxPush/Pop` range around the two kernels

Name the range "update\_and\_calc"

Include `<nvtx3/nvToolsExt.h>`

Update Makefile with `"-ldl"` at line 134

Replace `-k "..."` `ncu` command line with `"--nvtx --nvtx-include "update_and_calc/"` and `"--set full"` with `"--metrics ..."`

```
ncu --metrics sm_throughput.avg.pct_of_peak_sustained_elapsed --nvtx --nvtx-include
"update_and_calc/" -c 1 -f -o tea_leaf_%i ./tea_leaf
```

```
ncu -i tea_leaf_3.ncu-rep
```

```
[10768] tea_leaf@127.0.0.1
```

```
device_tea_leaf_ppcg_solve_update_r(kernel_info_t, ...), 2021-Aug-17 14:43:05, Context 1, Stream 7
```

```
NVTX Push/Pop Stack for Thread 10768:
```

```
<default domain>
```

```
<0,update_and_calc>
```

```
Section: Command line profiler metrics
```

```
sm_throughput.avg.pct_of_peak_sustained_elapsed    %    86.51
```

An abstract network visualization featuring a dark background with numerous glowing green nodes and thin, intersecting lines. The nodes are scattered across the frame, with some appearing as bright white-green points and others as softer, teal-colored circles. The lines connect these nodes in a complex, web-like pattern, creating a sense of interconnectedness and data flow.

**More: Data Collection**

# Collecting Data

By default, CLI results are printed to stdout

Use `--export/-o` to save results to a report file, use `-f` to force overwrite

```
$ ncu -f -o $HOME/my_report <app>  
$ my_report.ncu-rep
```

Use `--log-file` to pipe text output to a different stream (stdout/stderr/file)

Can use (env) variables available in your batch script or file macros to add report name placeholders

Full parity with `nvprof` filename placeholders/file macros

```
$ ncu -f -o $HOME/my_report_%h_${LSB_JOBID}_%p <app>  
$ my_report_host01_951697_123.ncu-rep
```

<https://docs.nvidia.com/nsight-compute/NsightComputeCli/index.html#command-line-options-file-macros>

# What To Collect

## Curated "sets" and "sections" with commonly-used, high-value metrics

```
$ ncu --list-sets
```

Identifier	Sections	Estimated Metrics
default	LaunchStats, Occupancy, SpeedOfLight	35
detailed	ComputeWorkloadAnalysis, InstructionStats, LaunchStats, MemoryWorkloadAnalysis, Occupancy, SchedulerStats, SourceCounters, SpeedOfLight, SpeedOfLight_RooflineChart, WarpStateStats	157
full	ComputeWorkloadAnalysis, InstructionStats, LaunchStats, MemoryWorkloadAnalysis, MemoryWorkloadAnalysis_Chart, MemoryWorkloadAnalysis_Tables, Occupancy, SchedulerStats, SourceCounters, SpeedOfLight, SpeedOfLight_RooflineChart, WarpStateStats	162
source	SourceCounters	47

## Use defaults, or combine as desired

```
$ ncu --set default --section SourceCounters --metrics sm__inst_executed_pipe_tensor.sum ./my-app
```

# What To Collect

## Query metrics for any targeted chip

```
$ ncu --query-metrics --chip gal100
smsp__warps_issue_stalled_not_selected          cumulative # of warps waiting
for the microscheduler to select the warp to issue
smsp__warps_issue_stalled_selected             cumulative # of warps selected
by the microscheduler to issue an instruction
smsp__warps_issue_stalled_short_scoreboard      cumulative # of warps waiting
for a scoreboard dependency on MIO operation other than (local, global, surface, tex)
...
tpc__cycles_active                             # of cycles where TPC was active
tpc__cycles_elapsed                            # of cycles where TPC was active
==PROF== Note that these metrics must be appended with a valid suffix before profiling them. See --help for
more information on --query-metrics-mode.
```

## Specify sub-metrics in section files, or on the command line

```
$ ncu --query-metrics-mode suffix --metrics sm__inst_executed_pipe_tensor ./my-app
sm__inst_executed_pipe_tensor.sum
sm__inst_executed_pipe_tensor.avg
sm__inst_executed_pipe_tensor.min
...
```

# Source Analysis

SASS (assembly) is always available, embedded into the report  
CUDA-C (Source) and PTX availability depends on compilation flags  
Use `-lineinfo` to include source/SASS correlation data in the binary

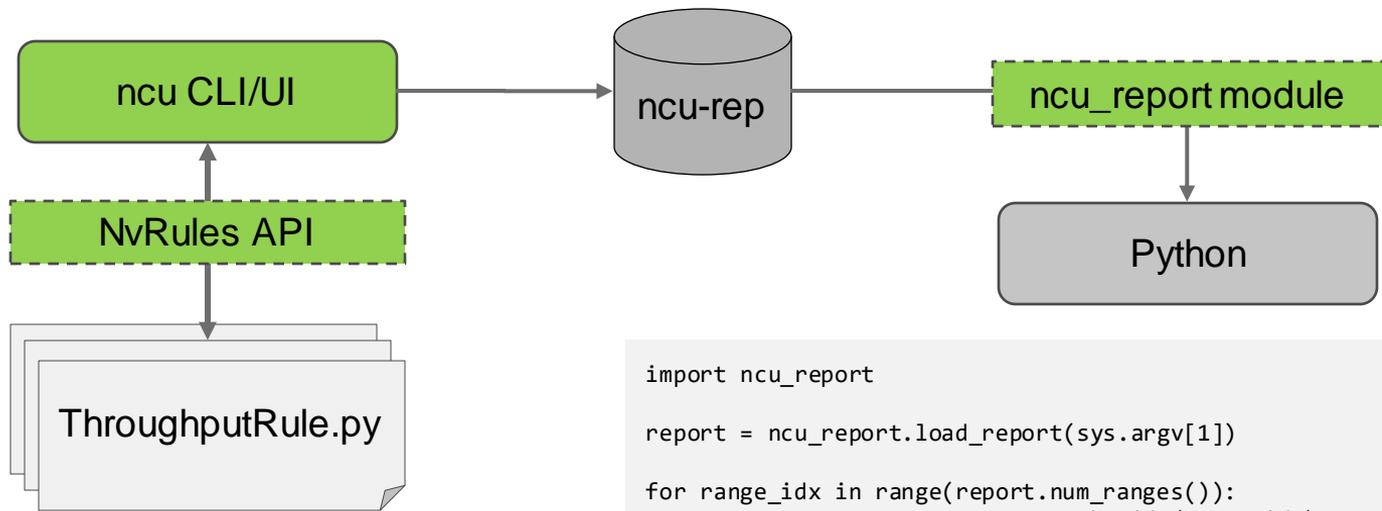
Source is not embedded in the report by default, need local or remote access to the source file to resolve in the UI. Import source during collection to (`--import-source yes`) to solve this.

Compiler optimizations can prevent exact source/SASS correlation

A network diagram with green nodes and lines on a dark background. The nodes are represented by small, glowing green circles of varying sizes, and they are interconnected by thin, light green lines. The overall appearance is that of a complex, interconnected network or data structure. The background is a dark, almost black, color with some subtle, larger-scale green and blue bokeh effects.

**More: Data Analysis**

# Python Interfaces



```
import ncu_report

report = ncu_report.load_report(sys.argv[1])

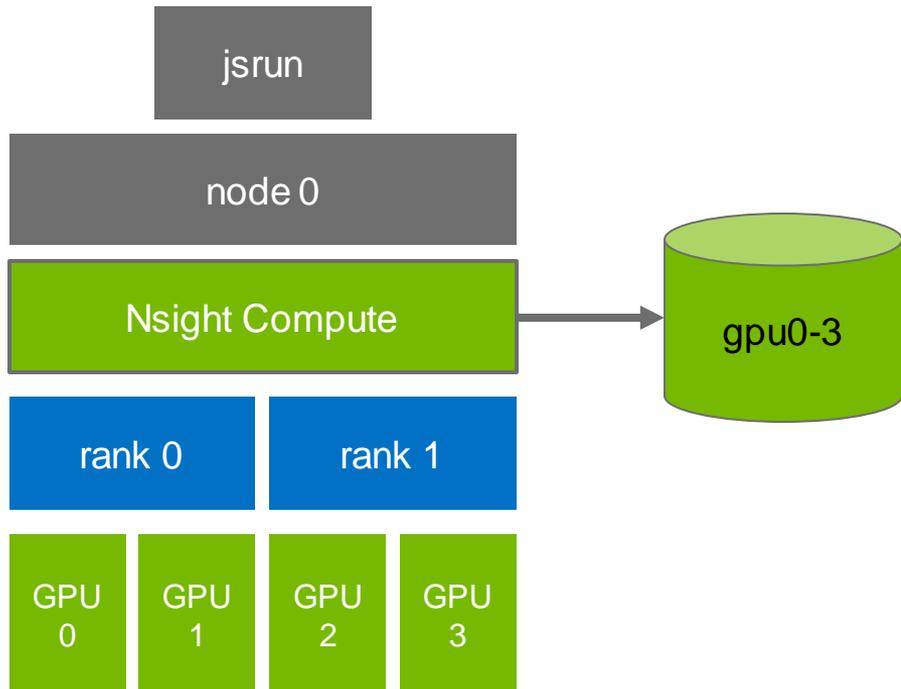
for range_idx in range(report.num_ranges()):
    current_range = report.range_by_idx(range_idx)
    for action_idx in current_range.actions_by_nvtx(["BottomRange/*/TopRange"], []):
        action = current_range.action_by_idx(action_idx)
        print(action.name())
        print(action.metric_by_name("gpc__cycles_elapsed.sum").as_uint64())
```

# Command Line Output

```
$ ncu -i /tmp/report.ncu-rep -c 2 --page raw --csv --metrics \  
gpc__cycles_elapsed.sum,sm__maximum_warps_per_active_cycle_pct
```

```
"ID","Process ID","Process Name","Host Name","Kernel Name","Kernel  
Time","Context","Stream","sm__maximum_warps_per_active_cycle_pct"  
"","","","","","","","","","%"  
"0","16301","tea_leaf","127.0.0.1","device_tea_leaf_ppcg_solve_update_r(kernel_info_t, double *, const  
double *, const double *, const double *)","2021-Dec-14 14:37:22","1","7","100.000000"  
"1","16301","tea_leaf","127.0.0.1","device_tea_leaf_ppcg_solve_calc_sd_new(kernel_info_t, const double  
*, double *, const double *, const double *, double *, const double *, const double *, const double *,  
const double *, const double *, const double *, const double *, int)","2021-Dec-14  
14:37:22","1","7","100.000000"
```

# Multi-Process Profiling

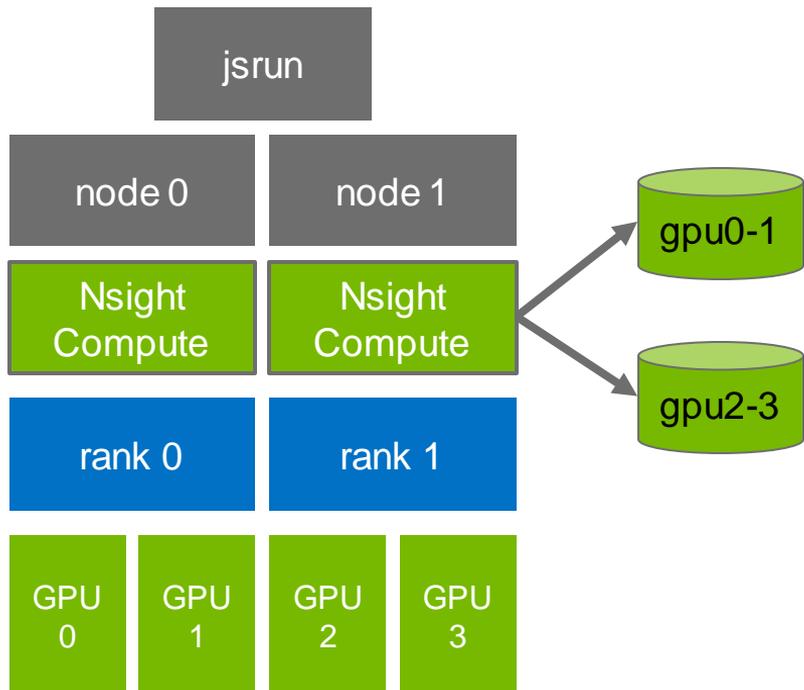


On a single-node submission, one Nsight Compute instance can profile all launched child processes

Data for all processes is stored in one report file

```
ncu --target-processes all -o <single-report-name> <app> <args>
```

# Multi-Process Profiling

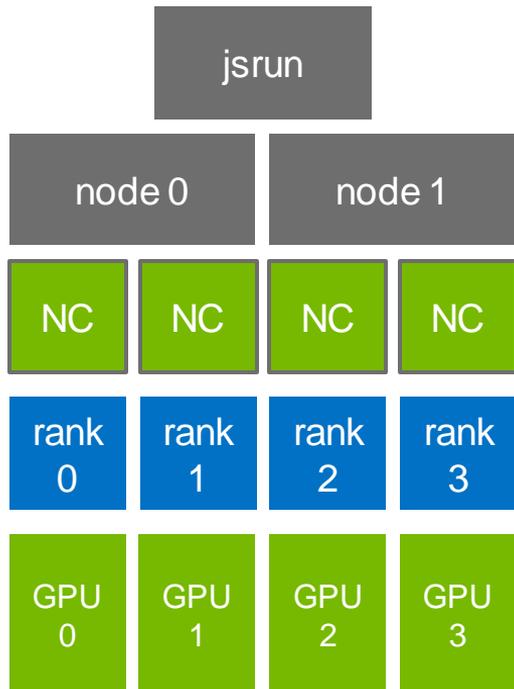


On multi-node submissions, one tool instance can be used per node

Ensure that instances don't write to the same report file on a shared disk

```
ncu -o report_%q{OMPI_COMM_WORLD_RANK}  
<app> <args>
```

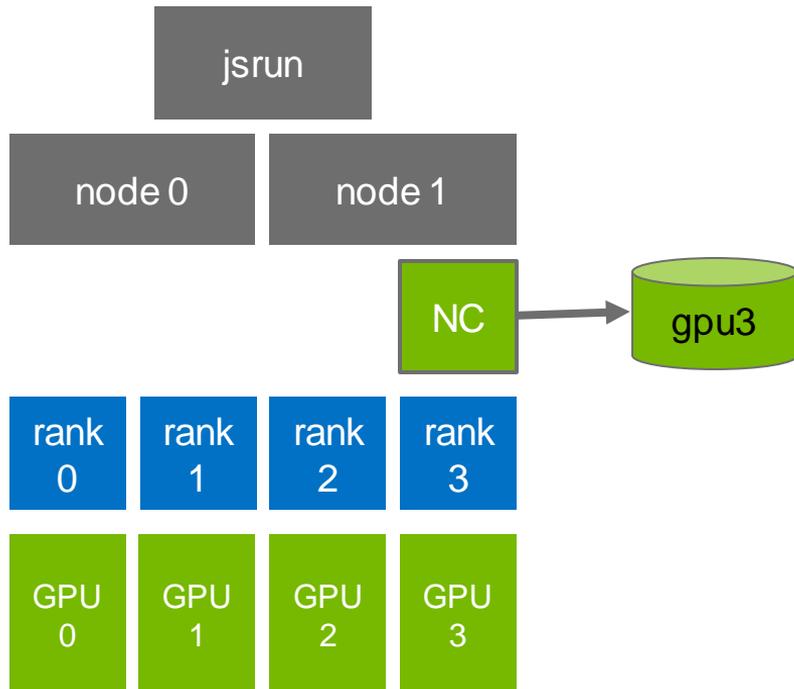
# Multi-Process Profiling



Multiple tool instances on the same node are supported, but...

All kernels across all GPUs will be serialized using system-wide file lock

# Multi-Process Profiling



Consider profiling only a single rank, e.g. using a wrapper script

```
#!/bin/bash
if [[ "$OMPI_COMM_WORLD_RANK" == "3" ]] ; then
    /sw/cluster/cuda/11.1/ nsight-compute/ncu -
o report_${OMPI_COMM_WORLD_RANK} --target-
processes all $*
else
    $*
fi
```



**Conclusion**

# Conclusion

Nsight Compute enables detailed CUDA kernel analysis

Rules give guidance on optimization opportunities and help metric understanding

Limit metrics to what is required when overhead is a concern. Consider using application replay.

Still requires level of hardware understanding to fully utilize the tool - pay attention to rule results and refer to <https://docs.nvidia.com/nsight-compute/ProfilingGuide/index.html>

Analyze results in the UI, or post-process with CSV output or python report interface

Check known issues: <https://docs.nvidia.com/nsight-compute/ReleaseNotes/index.html#known-issues>

# Further Reading

- Download** <https://developer.nvidia.com/nsight-compute> (can be newer than toolkit version)
- Documentation** <https://docs.nvidia.com/nsight-compute> (and local with the tool)  
<https://docs.nvidia.com/nsight-compute/ProfilingGuide/index.html>
- Forums** <https://devtalk.nvidia.com>
- Further Reading** <https://developer.nvidia.com/nsight-compute-videos>  
<https://developer.nvidia.com/nsight-compute-blogs>
- <https://github.com/NVIDIA/nsight-training>  
Repository with interactive training material for multiple Nsight tools, including Systems and Compute.
- <https://gitlab.com/NERSC/roofline-on-nvidia-gpus>



**NVIDIA**®