



Nsight Compute

Tools and techniques for making efficient use of GPUs

Felix Schmitt

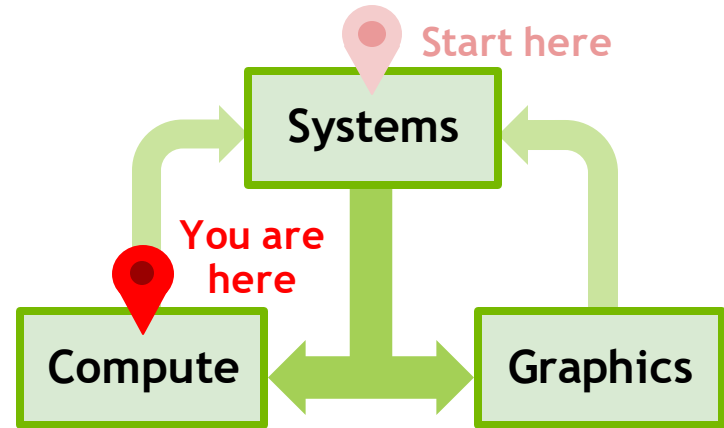
Nsight Product Family

Workflow

Nsight Systems - Analyze application algorithms system-wide

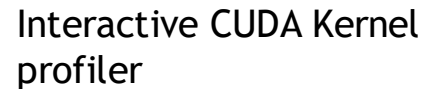
Nsight Compute - Analyze CUDA kernels

Nsight Graphics - Debug/analyze graphics workloads



The background of the slide is a dark, almost black, field. It is populated with numerous thin, light green lines that crisscross the space in various directions. Interspersed among these lines are several small, bright green circular dots. Some of these dots are slightly larger and more prominent than others. The overall effect is one of a complex, interconnected network or a starry sky with artificial elements.

Overview



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

API Stream

15341 > device_tea_leaf_ppcg_solve_update.r

Next Trigger: nvtx_tea_leaf_ppcg_solve_calc(update)

ID	API Name	Details	Func Return	Func Parameter
43833	cudaMalloc		CUDA_SUCCESS(0)	(0x55d77b64830f, 0x55d77b64830f)
43834	cuMemAlloc_v2		CUDA_SUCCESS(0)	(0x55d77b64830f, 0x55d77b64830f)
43835	cudaMalloc		CUDA_SUCCESS(0)	(0x55d77b64830f, 0x55d77b64830f)
43836	cuMemAlloc_v2		CUDA_SUCCESS(0)	(0x55d77b64830f, 0x55d77b64830f)
43837	cudaMemcpy		CUDA_SUCCESS(0)	(0x7feb89f6cc00, 0x7feb89f6cc00, 0x7feb89f6cc00, 0x7feb89f6cc00, 0x7feb89f6cc00, 0x7feb89f6cc00)
43838	cuMemcpyHtoD_v2		CUDA_SUCCESS(0)	(0x7feb89f6cc00, 0x7feb89f6cc00, 0x7feb89f6cc00, 0x7feb89f6cc00, 0x7feb89f6cc00, 0x7feb89f6cc00)
43839	cudaMemcpy		CUDA_SUCCESS(0)	(0x7feb89f6cc00, 0x7feb89f6cc00, 0x7feb89f6cc00, 0x7feb89f6cc00, 0x7feb89f6cc00, 0x7feb89f6cc00)
43840	cuMemcpyHtoD_v2		CUDA_SUCCESS(0)	(0x7feb89f6cc00, 0x7feb89f6cc00, 0x7feb89f6cc00, 0x7feb89f6cc00, 0x7feb89f6cc00, 0x7feb89f6cc00)
43841	cudaLaunchKernel		CUDA_SUCCESS(0)	(0x55d77b64830f, 0x55d77b64830f, 0x55d77b64830f, 0x55d77b64830f, 0x55d77b64830f, 0x55d77b64830f)
43842	cuLaunchKernel		CUDA_SUCCESS(0)	(0x55d77b64830f, 0x55d77b64830f, 0x55d77b64830f, 0x55d77b64830f, 0x55d77b64830f, 0x55d77b64830f)
43843	device_tea_leaf_ppcg_solve_init	device_tea_leaf_ppcg_solve_init	CUDA_SUCCESS(0)	(0x55d77b64830f, 0x55d77b64830f, 0x55d77b64830f, 0x55d77b64830f, 0x55d77b64830f, 0x55d77b64830f)
43844	cuLaunchKernel		CUDA_SUCCESS(0)	(0x55d77b64830f, 0x55d77b64830f, 0x55d77b64830f, 0x55d77b64830f, 0x55d77b64830f, 0x55d77b64830f)
43845	cuLaunchKernel		CUDA_SUCCESS(0)	(0x55d77b64830f, 0x55d77b64830f, 0x55d77b64830f, 0x55d77b64830f, 0x55d77b64830f, 0x55d77b64830f)
43846	reduction	reduction	CUDA_SUCCESS(0)	(0x55d77b64830f, 0x55d77b64830f, 0x55d77b64830f, 0x55d77b64830f, 0x55d77b64830f, 0x55d77b64830f)
43847	cuLaunchKernel		CUDA_SUCCESS(0)	(0x55d77b64830f, 0x55d77b64830f, 0x55d77b64830f, 0x55d77b64830f, 0x55d77b64830f, 0x55d77b64830f)
43848	cuLaunchKernel		CUDA_SUCCESS(0)	(0x55d77b64830f, 0x55d77b64830f, 0x55d77b64830f, 0x55d77b64830f, 0x55d77b64830f, 0x55d77b64830f)
43849	reduction	reduction	CUDA_SUCCESS(0)	(0x55d77b64830f, 0x55d77b64830f, 0x55d77b64830f, 0x55d77b64830f, 0x55d77b64830f, 0x55d77b64830f)
43850	cuLaunchKernel		CUDA_SUCCESS(0)	(0x55d77b64830f, 0x55d77b64830f, 0x55d77b64830f, 0x55d77b64830f, 0x55d77b64830f, 0x55d77b64830f)
43851	cuLaunchKernel		CUDA_SUCCESS(0)	(0x55d77b64830f, 0x55d77b64830f, 0x55d77b64830f, 0x55d77b64830f, 0x55d77b64830f, 0x55d77b64830f)
43852	reduction	reduction	CUDA_SUCCESS(0)	(0x55d77b64830f, 0x55d77b64830f, 0x55d77b64830f, 0x55d77b64830f, 0x55d77b64830f, 0x55d77b64830f)
43853	cudaMemcpy		CUDA_SUCCESS(0)	(0x7feb89f6cc00, 0x7feb89f6cc00, 0x7feb89f6cc00, 0x7feb89f6cc00, 0x7feb89f6cc00, 0x7feb89f6cc00)
43854	cuMemcpyHtoD_v2		CUDA_SUCCESS(0)	(0x7feb89f6cc00, 0x7feb89f6cc00, 0x7feb89f6cc00, 0x7feb89f6cc00, 0x7feb89f6cc00, 0x7feb89f6cc00)
43855	cuLaunchKernel		CUDA_SUCCESS(0)	(0x55d77b64830f, 0x55d77b64830f, 0x55d77b64830f, 0x55d77b64830f, 0x55d77b64830f, 0x55d77b64830f)
43856	cuLaunchKernel		CUDA_SUCCESS(0)	(0x55d77b64830f, 0x55d77b64830f, 0x55d77b64830f, 0x55d77b64830f, 0x55d77b64830f, 0x55d77b64830f)
43857	device_tea_leaf_ppcg_solve_init...	device_tea_leaf_ppcg_solve_init_sd_new	CUDA_SUCCESS(0)	(0x55d77b64830f, 0x55d77b64830f, 0x55d77b64830f, 0x55d77b64830f, 0x55d77b64830f, 0x55d77b64830f)
43858	cuLaunchKernel		CUDA_SUCCESS(0)	(0x55d77b64830f, 0x55d77b64830f, 0x55d77b64830f, 0x55d77b64830f, 0x55d77b64830f, 0x55d77b64830f)
43859	cuLaunchKernel		CUDA_SUCCESS(0)	(0x55d77b64830f, 0x55d77b64830f, 0x55d77b64830f, 0x55d77b64830f, 0x55d77b64830f, 0x55d77b64830f)

Resources

CUDA Memory Allocations

Export to SVG Export to GraphViz

Enter filter, e.g. \$[ID] >= 10

ID	API Call ID	Allocation type	Address	Size Requested	Context	Device ID
Total allocations: 41						
Total size: 2.63 Gbytes						
1	603	UNIFIED MEMORY ALLOC	0x7fec32000000	32 bytes	0x55d77b64830f	0
2	655	DEVICE MEMORY ALLOC	0x7fec24000000	122.19 Mbytes	0x55d77b64830f	0
3	666	DEVICE MEMORY ALLOC	0x7fec22000000	122.19 Mbytes	0x55d77b64830f	0
4	677	DEVICE MEMORY ALLOC	0x7fec14000000	122.19 Mbytes	0x55d77b64830f	0
5	688	DEVICE MEMORY ALLOC	0x7fec12000000	122.19 Mbytes	0x55d77b64830f	0

Sections/Rules Info API Statistics NVTX Resources CPU Call Stack

(Non-interactive) Profile

Activity

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_%.
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_11_3-x64/ncu - export /tmp/var/report_%. --force-override --target-processes application-only --replay-mode kernel --kernel-name-base function --launch-skip-before-match 0 --launch

Cancel Reset Activity Launch

Command Line

```
$-/working/git/TeaLeaf_CUDA5 /tmp/var/target/linux-desktop-glibc_2_11_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 0.0000000 timestep 4.00E-03  
Switching after 990 CG its, error 0.9911110E+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 [%] % 91.70  
DRAM Throughput % 91.70  
Duration msecond 1.00  
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

Current

Launch

Time

Cycles

Regs

GPU

SM Frequency

CC

Process

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-...	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_solve_init
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	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 offline analysis.

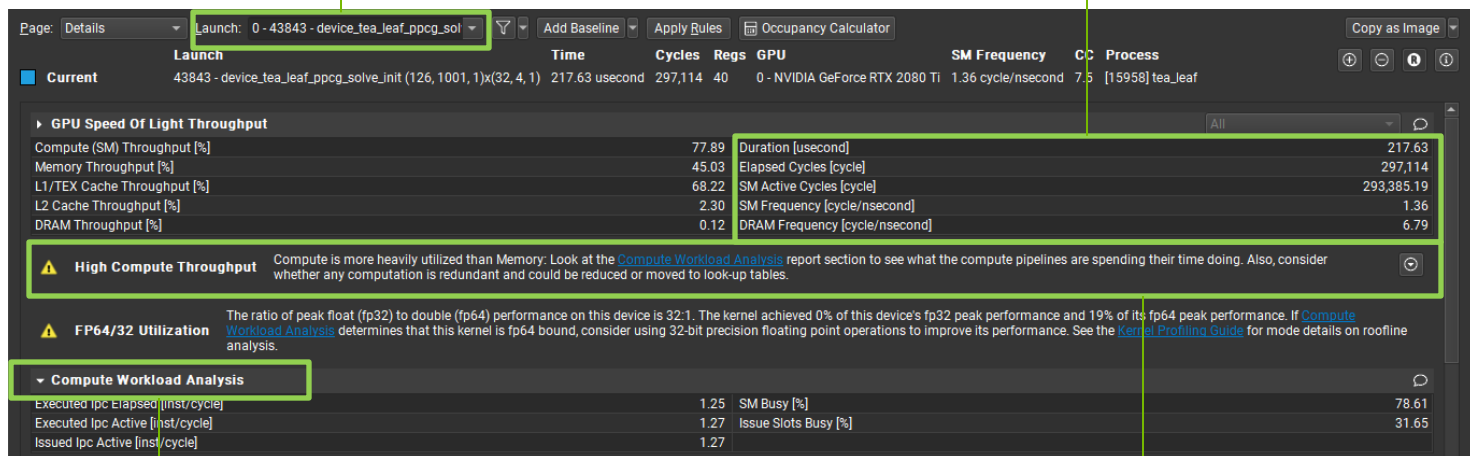
Compute Workload Analysis

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

Profiler Report

Selected result

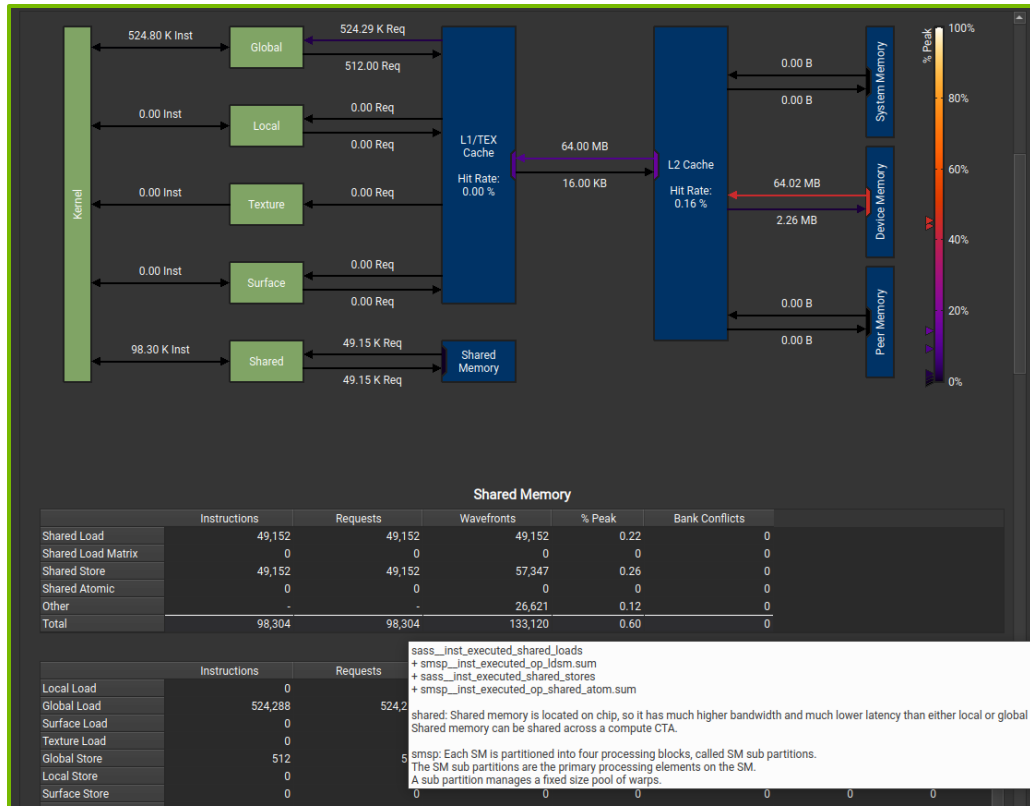
Metric values



Expandable Sections

Expert Analysis (Rules)

Profiler Report

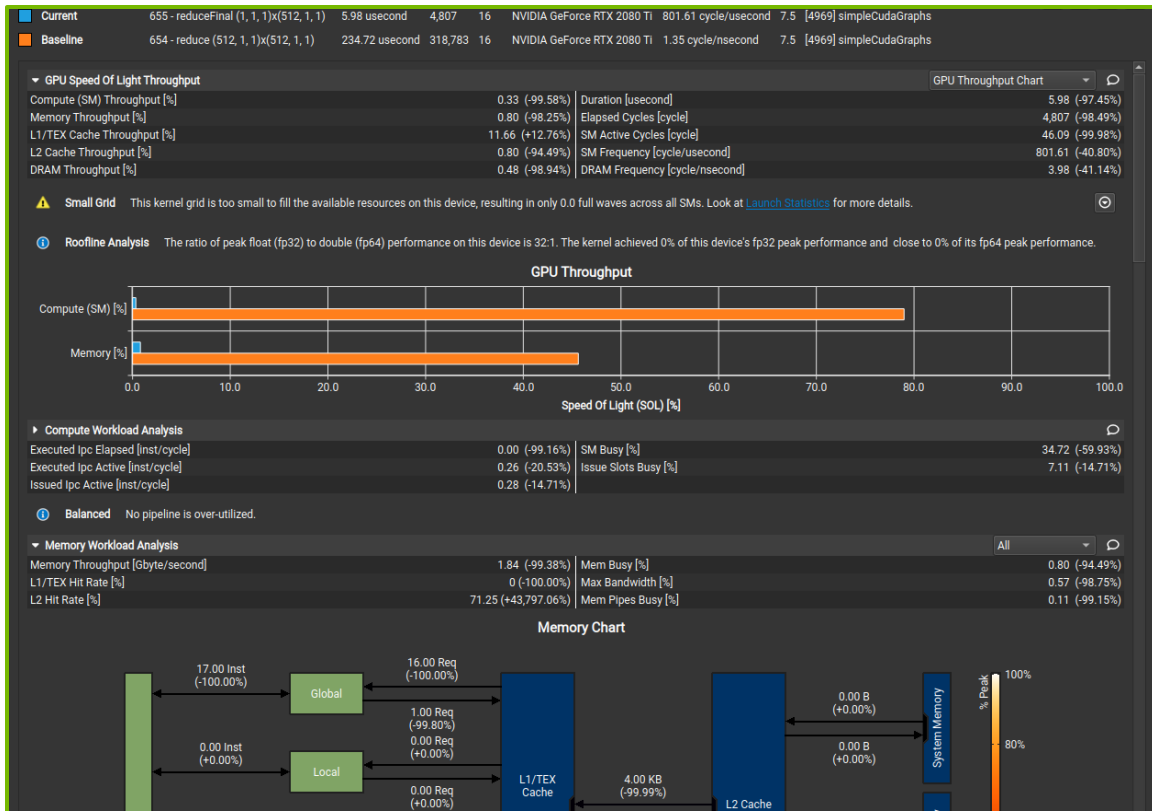


Detailed memory workload analysis chart and tables

Shows transferred data or throughputs

Tooltips provide metric names, calculation formulas and detailed background info

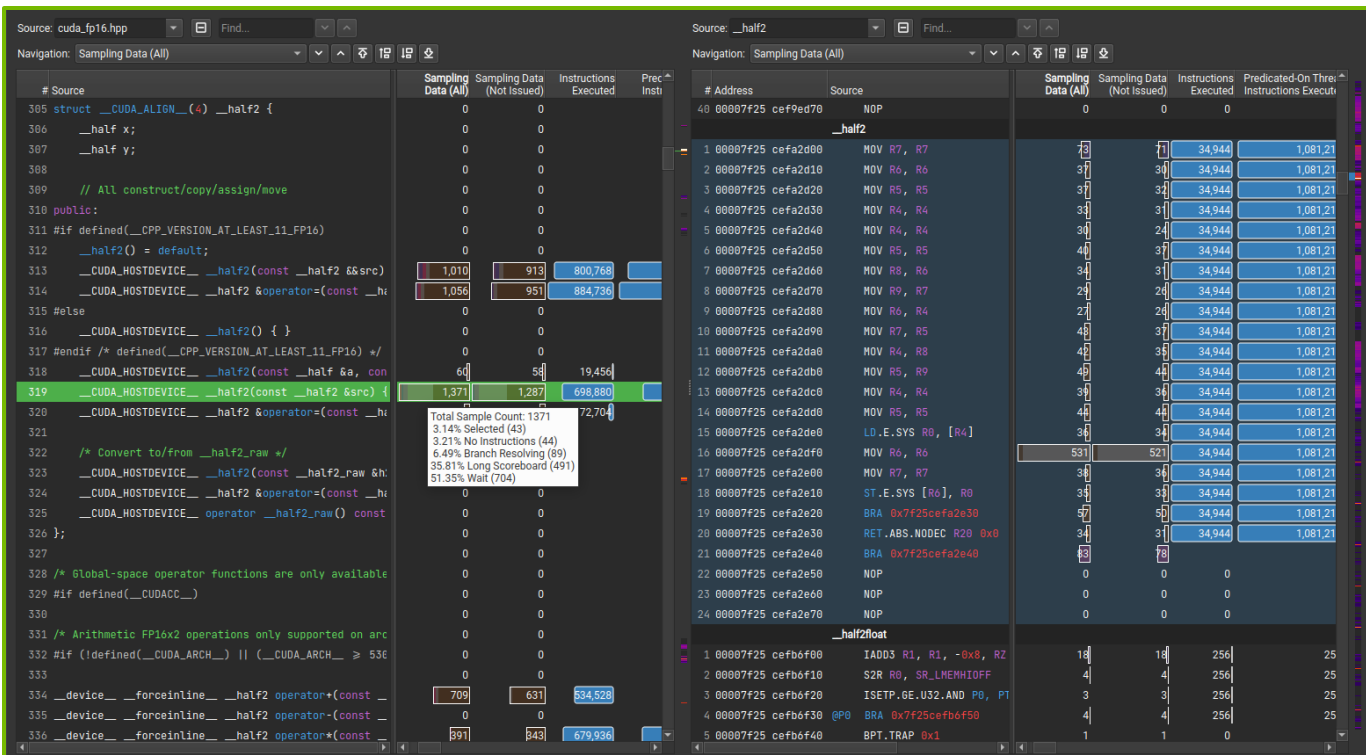
Profiler Report



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: 7.5

Threads Per Block: 128

Shared Memory Size Config (bytes): 65536

Registers Per Thread: 40

CUDA version: 11.0

Shared Memory Per Block (bytes): 3072

Global Load Cache Mode: L1+L2 (ca)

Apply

Reset

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 %

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

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		

The occupancy is limited by block size

11



Command Line Collection and Analysis

Example 1: collecting the "full" sections set for 10 instances of specific kernels, writing results to a uniquely named report

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

Example 2: printing selected metrics for two kernels in raw CSV format from a pre-collected report

```
$ 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"
```

The background of the slide is a dark, almost black, field. It is populated with numerous thin, light green lines that crisscross the frame in various directions. Interspersed among these lines are several small, bright green circular dots or nodes. Some of these dots appear slightly larger or more intense than others. The overall effect is a complex, web-like pattern that suggests a network or a data structure. In the lower right corner, the text 'Hands-On with TeaLeaf_Cuda' is displayed in a clean, white, sans-serif font. The text is arranged in two lines, with 'Hands-On with' on the top line and 'TeaLeaf_Cuda' on the bottom line. The white text provides a sharp contrast against the dark background.

Hands-On with TeaLeaf_Cuda

The Application

Get the source code

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

Get compiler, (Open)MPI and CUDA. Data collected on GA100 with:

CUDA 11.5, OpenMPI, Nsight-Compute/2022.1.1

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

Instrument with NVTX (as necessary)

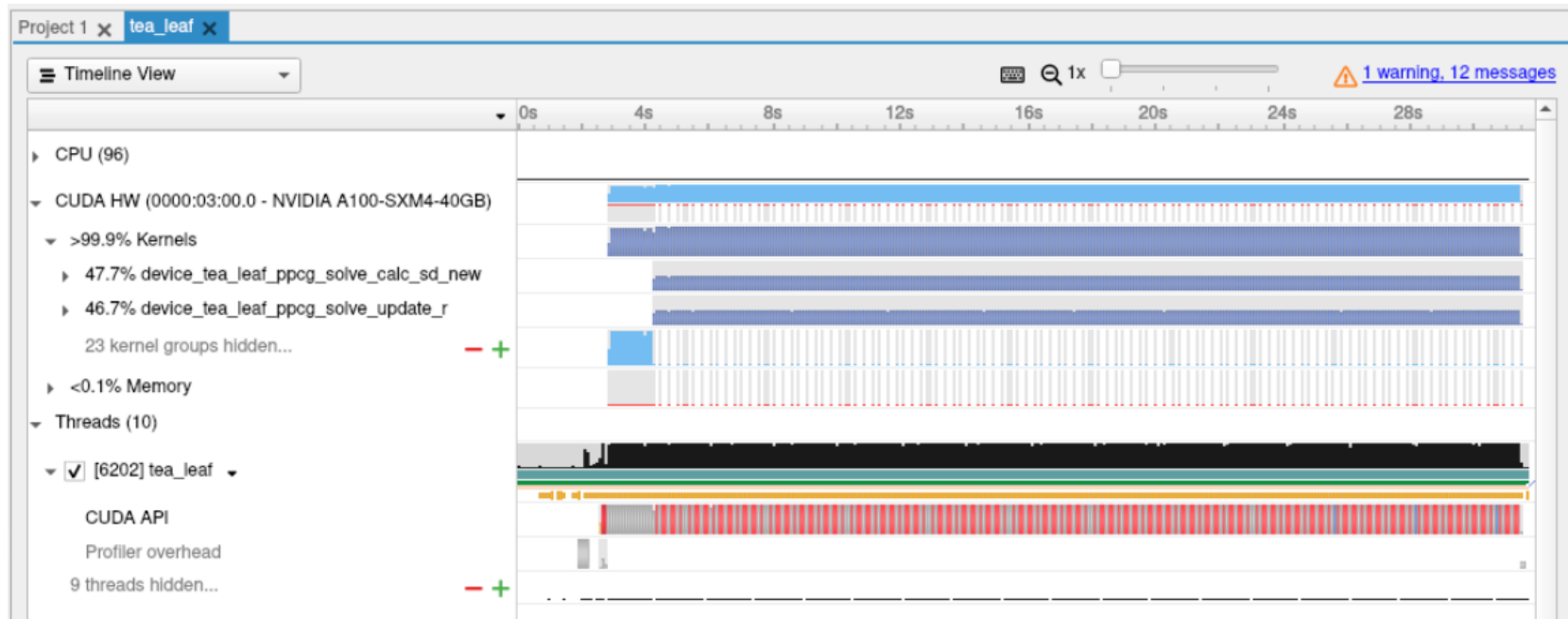
```
nvtxRangePush("update_and_calc");
nvtxRangePush("update");
device_tea_leaf_ppcg_solve_update_r
<<<matrix_power_grid_dim, block_shape>>>
[...]
nvtxRangePop();

nvtxRangePush("calc");
device_tea_leaf_ppcg_solve_calc_sd_new
<<<matrix_power_grid_dim, block_shape>>>
[...]
nvtxRangePop();
nvtxRangePop();
```


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



Interactive Profiling Example

Live

Profiling with Nsight Compute

Collected 10 instances of these kernels with ncu command line, "full" set of metrics. Inspect the resulting report in the Nsight Compute UI (ncu-ui).

Follow the rule links and guidance for best experience.

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

Page: Summary Result: 0 - 32940 - device_tea_leaf_ppcg_solve... Add Baseline Apply Rules Occupancy Calculator Copy as Image

Result Time Cycles Regs GPU SM Frequency CC Process

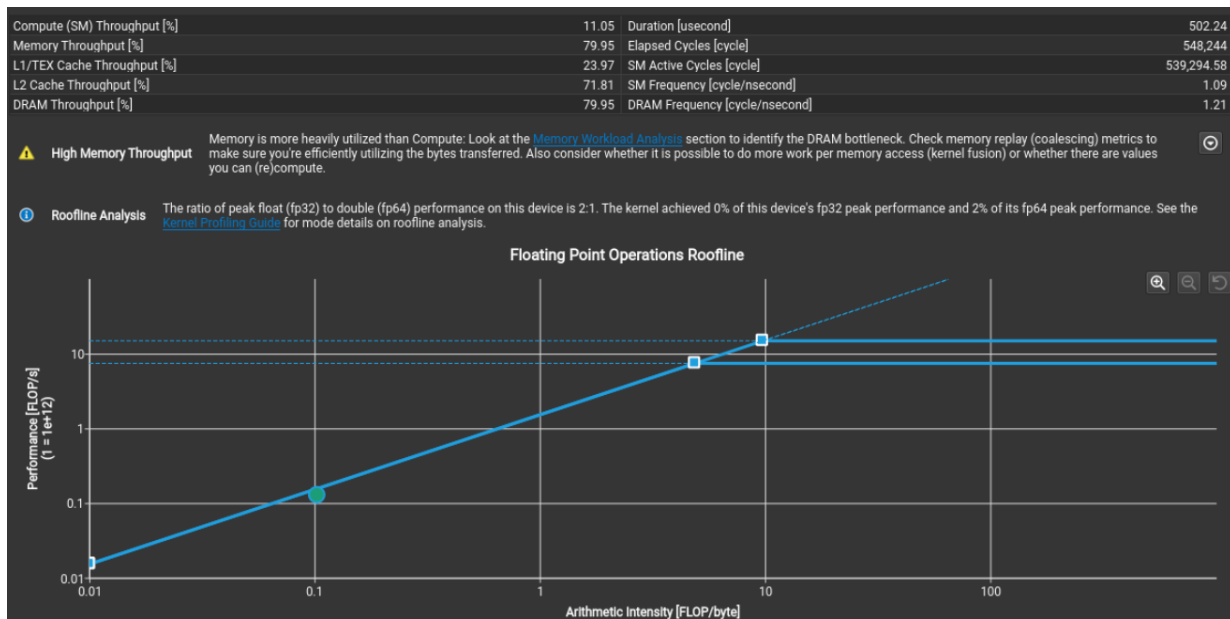
Current 32940 - device_tea_leaf_ppcg_solve_update_r (126, 1001, 1)x(32, 4, 1) 475.90 usecond 512,011 32 0 - NVIDIA A100-SXM4-40GB 1.08 cycle/nsecond 8.0 [9073] tea_leaf

Use the column headers to sort the results in this report. Double-click a result to see detailed metrics.

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

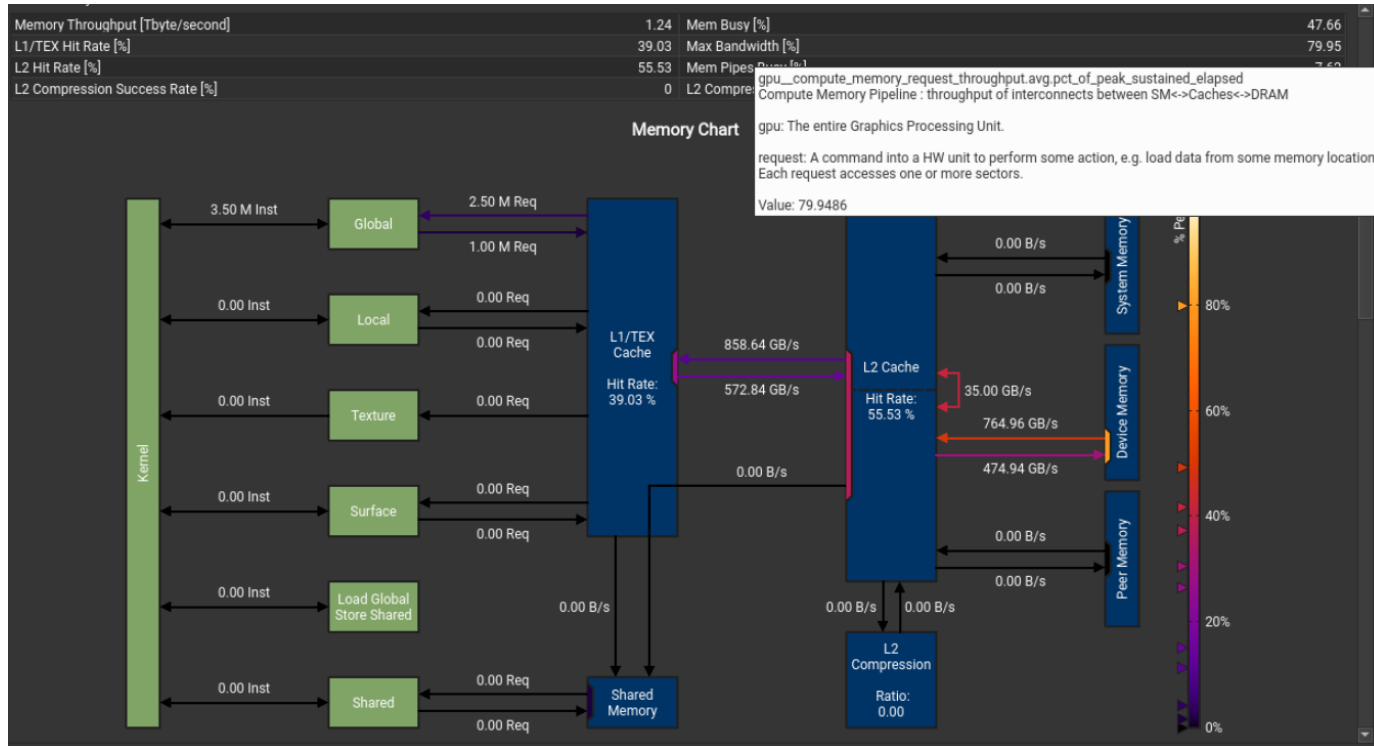
Analyzing calc kernel

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)



Analyzing calc kernel

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

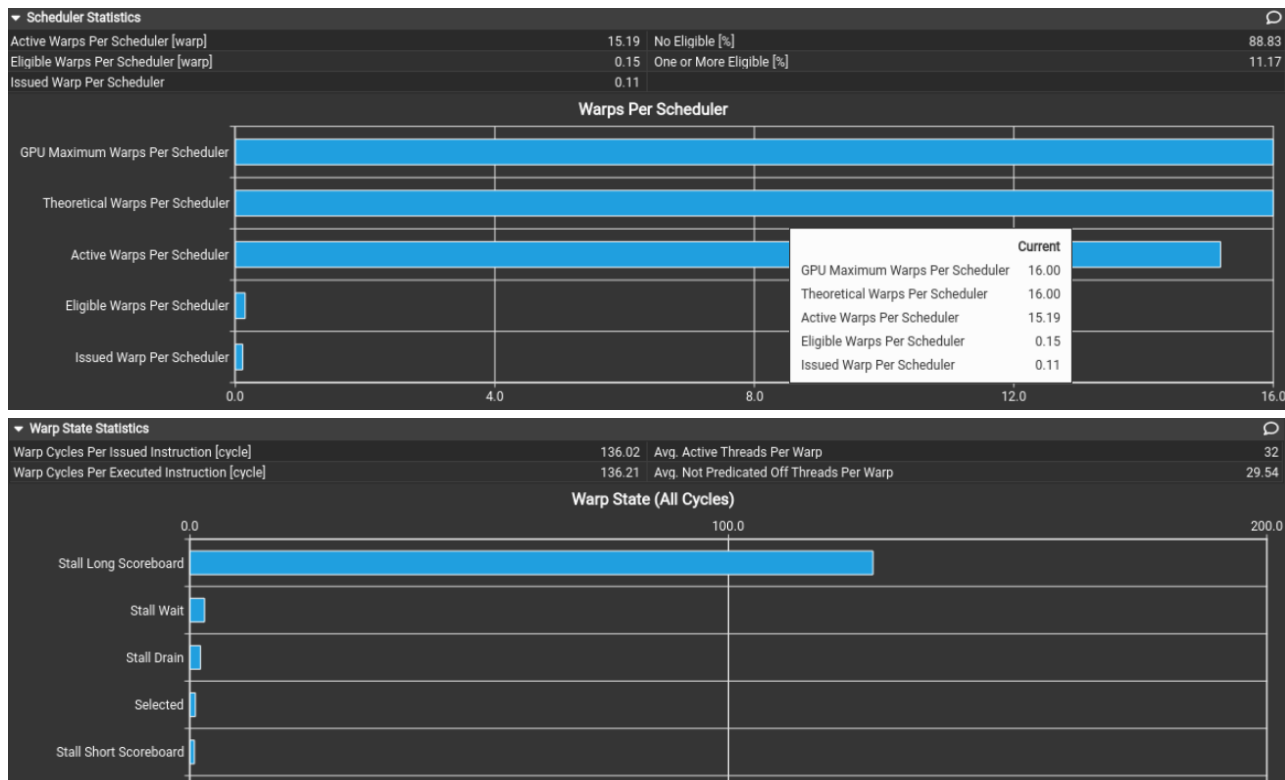


Analyzing calc kernel

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 (%)
0x14ff32fbfe20 in device_tea_leaf_ppcg_solve_calc_sd_new	500,000	20
0x14ff32fbfe00 in device_tea_leaf_ppcg_solve_calc_sd_new	500,000	20
0x14ff32fbfd0 in device_tea_leaf_ppcg_solve_calc_sd_new	500,000	20
0x14ff32fbfdb0 in device_tea_leaf_ppcg_solve_calc_sd_new	500,000	20
0x14ff32fbfda0 in device_tea_leaf_ppcg_solve_calc_sd_new	500,000	20

Warp Stall Sampling (All Cycles)

Location	Value	Value (%)
0x14ff32fbfd00 in device_tea_leaf_ppcg_solve_calc_sd_new	26,623	47
0x14ff32fbfe10 in device_tea_leaf_ppcg_solve_calc_sd_new	26,144	46
0x14ff32fbfe40 in device_tea_leaf_ppcg_solve_calc_sd_new	1,407	9
0x14ff32fbfd00 in device_tea_leaf_ppcg_solve_calc_sd_new	664	1
0x14ff32fbfb80 in device_tea_leaf_ppcg_solve_calc_sd_new	334	1

Analyzing calc kernel

Stalled at DMUL instruction, waiting for LDG (load global) in line 43 (via register R10)

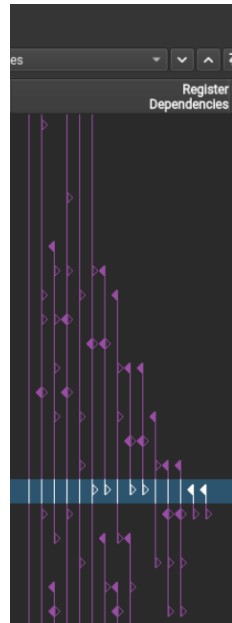
LDG instructions are uncoalesced

Lots of excessive (non-ideal) L2 sector accesses

View: SASS

Source: device_tea_leaf_ppcg_solve_calc_sd_new Find... Navigation: Warp Stall Sampling (All Cycles)

#	Address	Source	Live Registers	Warp Stall Sampling (All Cycles)	Warp Stall Sampling (Not-issued Cycles)	Instructions Executed	L2 Theoretical Sectors Global Excessive	L2 Theoretical Sectors Global
30	000014ff	32fbfcd0 ISETP.6T.0R P0, PT, R3, UR5, P0	3	56	24	504,504		
31	000014ff	32fbfce0 @P0 EXIT	3	135	55	504,504		
32	000014ff	32fbfcf0 ULDC UR4, c[0x0][0x164]	3	48	20	500,000		
33	000014ff	32fbfd00 MOV R2, c[0x0][0x1f0]	4	16	0	500,000		
34	000014ff	32fbfd10 ULEA UR4, UR7, UR4, 0x1	4	5	0	500,000		
35	000014ff	32fbfd20 IMAD.MOV.U32 R15, R2, R2, 0x8	5	6	0	500,000		
36	000014ff	32fbfd30 IMAD.WIDE R8, R15, R2, c[0x0][0x1e0]	7	42	15	500,000		
37	000014ff	32fbfd40 IMAD R0, R4, UR4, R3	8	39	14	500,000		
38	000014ff	32fbfd50 IMAD.WIDE R2, R0, R15, c[0x0][0x1e0]	7	42	2	500,000		
39	000014ff	32fbfd60 LDG.E.64.CONSTANT R8, [R8.64]	7	75	60	500,000	0	500,000
40	000014ff	32fbfd70 IMAD.WIDE R10, R0, R15, c[0x0][0x1a8]	9	25	9	500,000		
41	000014ff	32fbfd80 LDG.E.64.CONSTANT R2, [R2.64]	9	56	26	500,000	0	500,000
42	000014ff	32fbfd90 IMAD.WIDE R4, R0, R15, c[0x0][0x190]	11	28	8	500,000		
43	000014ff	32fbfda0 LDG.E.64.CONSTANT R10, [R10.64]	11	64	24	500,000	500,000	4,500,000
44	000014ff	32fbfdb0 LDG.E.64 R6, [R4.64]	13	72	48	500,000	500,000	4,500,000
45	000014ff	32fbfdc0 DMUL R12, R10, R8	15	26,623	23,471	500,000		
46	000014ff	32fbfdd0 DFMA R6, R6, R2, R12	11	66	59	500,000		
47	000014ff	32fbfde0 IMAD.WIDE R12, R0, R15, c[0x0][0x1b0]	9	13	6	500,000		
48	000014ff	32fbfdf0 STG.E.64 [R4.64], R6	7	75	60	500,000	500,000	4,500,000
49	000014ff	32fbfe00 LDG.E.64 R14, [R12.64]	7	43	31	500,000	500,000	4,500,000
50	000014ff	32fbfe10 DADD R14, R6, R14	7	26,144	23,172	500,000		



Analyzing calc kernel

Where is this in the code?

Need to add *-lineinfo* flag
in Makefile during compilation
(NV_FLAGS) for CUDA-C/SASS
correlation.

Re-compile, re-run

Consider using
--import-source yes

The screenshot displays the NVIDIA Nsight Compute interface, showing the correlation between C++ source code and SASS instructions for a 'calc' kernel. The interface is divided into three main panes:

- Left Pane (Source Code):** Shows the C++ source code for the 'calc' kernel. The code includes a loop over 'step' and 'rtemp' variables, with a conditional execution based on 'PRECONDITIONER' and 'WITHIN_BOUNDS'. The code is annotated with line numbers 231 to 250.
- Middle Pane (SASS Instructions):** Shows the SASS instructions corresponding to the source code. The instructions are listed with their addresses and cycle counts. The instructions include 'IMAD', 'LDG.E.64', 'DMUL', 'DFMA', 'DADD', 'STG.E.64', 'ULDC', and 'UIADD3'. The instructions are annotated with line numbers 37 to 56.
- Right Pane (SASS Instructions):** Shows the SASS instructions corresponding to the source code. The instructions are listed with their addresses and cycle counts. The instructions include 'IMAD', 'LDG.E.64', 'DMUL', 'DFMA', 'DADD', 'STG.E.64', 'ULDC', and 'UIADD3'. The instructions are annotated with line numbers 44 to 56.

The background of the slide is a dark, almost black, space filled with a complex network of thin, light green lines. These lines connect various points, creating a web-like structure. The points themselves are small, glowing green circles of varying sizes and brightness. Some points are more prominent, appearing as larger, brighter green spheres, while others are smaller and dimmer. The overall effect is one of a dynamic, interconnected system, possibly representing a data network or a complex process.

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
sm__warps_issue_stalled_not_selected          cumulative # of warps waiting
for the microscheduler to select the warp to issue
sm__warps_issue_stalled_selected              cumulative # of warps selected
by the microscheduler to issue an instruction
sm__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

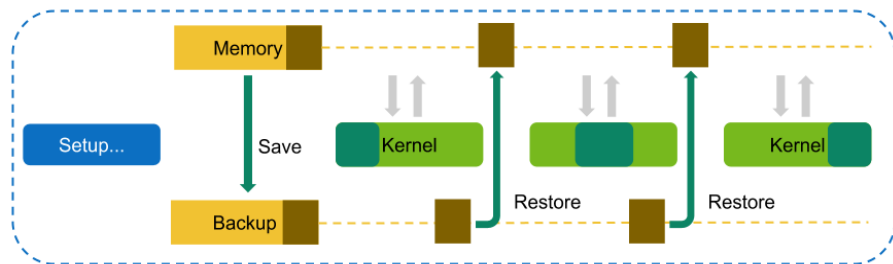
SASS (assembly) is always available, embedded into the report
CUDA-C (Source) and PTX availability depends on compilation flags
Use -lineinfo to include source/SASS correlation data in the binary

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.

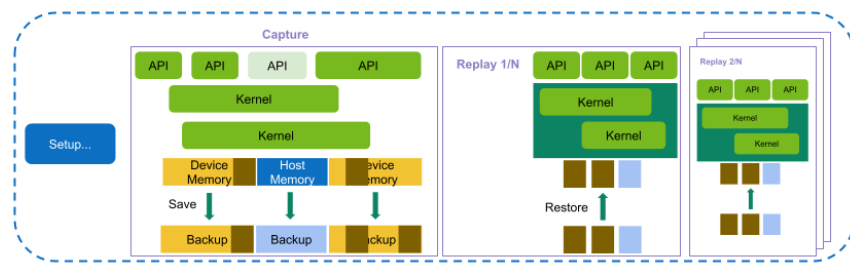
Compiler optimizations can prevent exact source/SASS correlation

Replay Modes

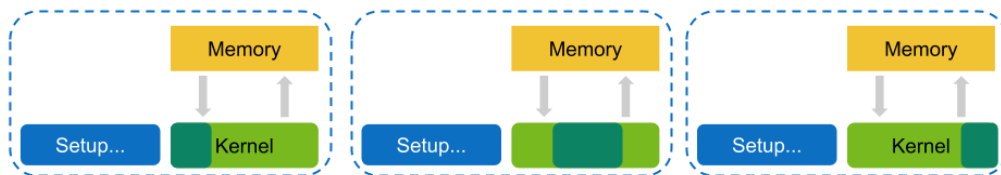
<https://docs.nvidia.com/nsight-compute/ProfilingGuide/index.html#replay>



Kernel Replay
(interactive and non-interactive)



Range Replay
(non-interactive)



Application Replay
(non-interactive)

The background of the slide is a dark, almost black, field. It is populated with numerous thin, light green lines that crisscross the space in various directions. Interspersed among these lines are several small, bright green circular dots. Some of these dots are slightly larger and more prominent than others. The overall effect is one of a complex, interconnected network or a starry sky with artificial elements.

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

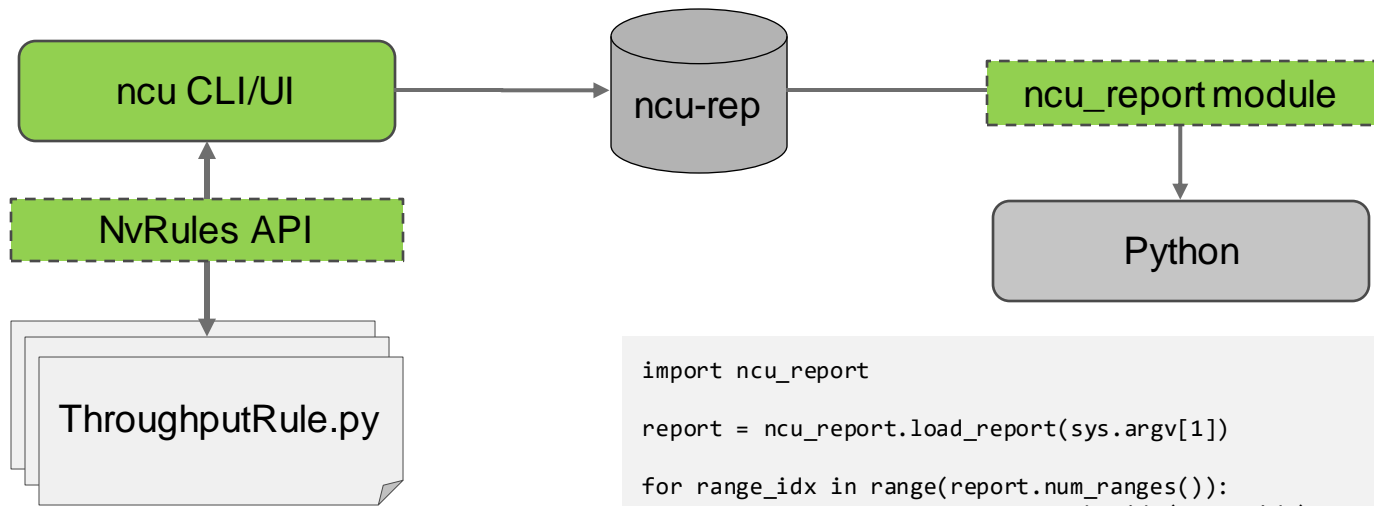
Check known issues: <https://docs.nvidia.com/nsight-compute/ReleaseNotes/index.html#known-issues>

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



Python Interfaces

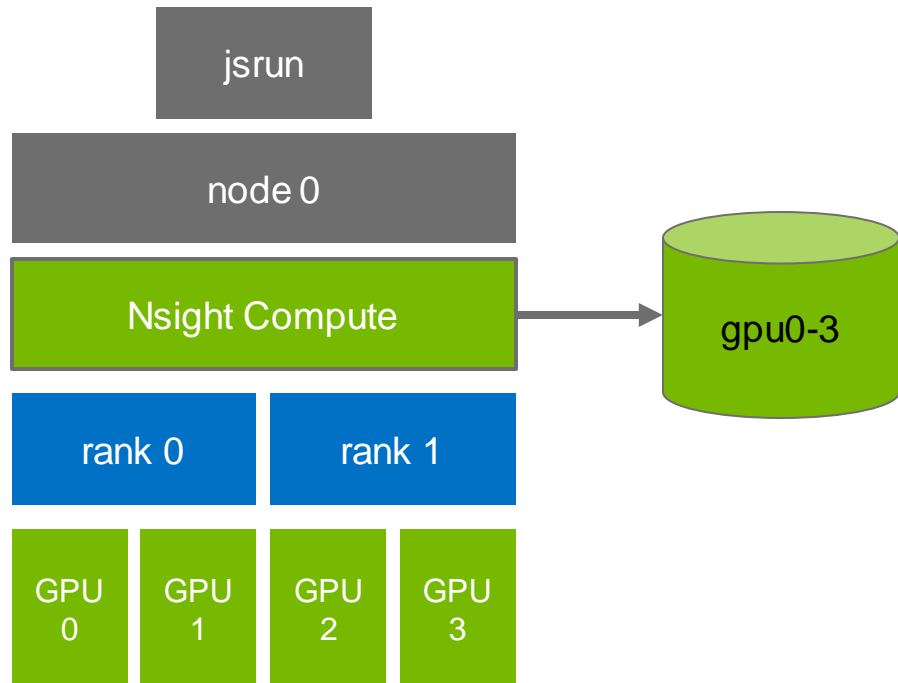


```
import ncu_report

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())
```

Multi-Process Profiling

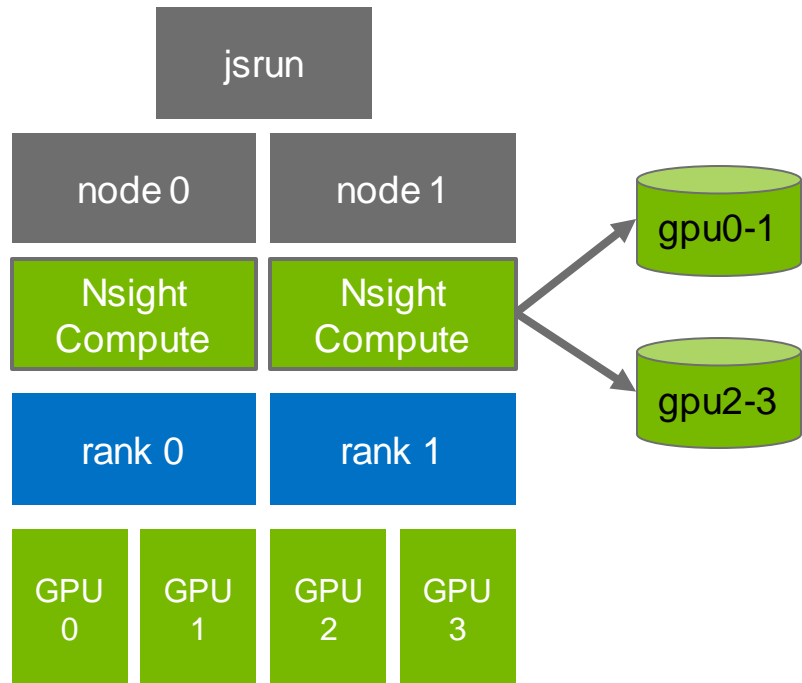


On a single-node submission, one Nsight Compute instance can profile all launched child processes

Data for all processes is stored in one report file

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

Multi-Process Profiling

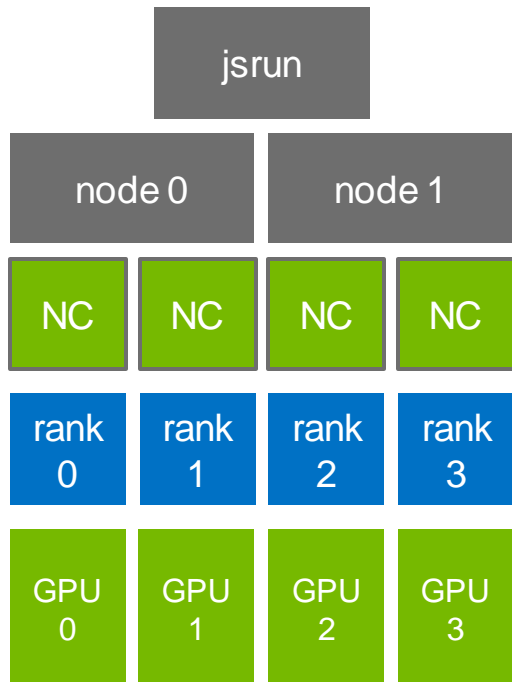


On multi-node submissions, one tool instance can be used per node

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>
```

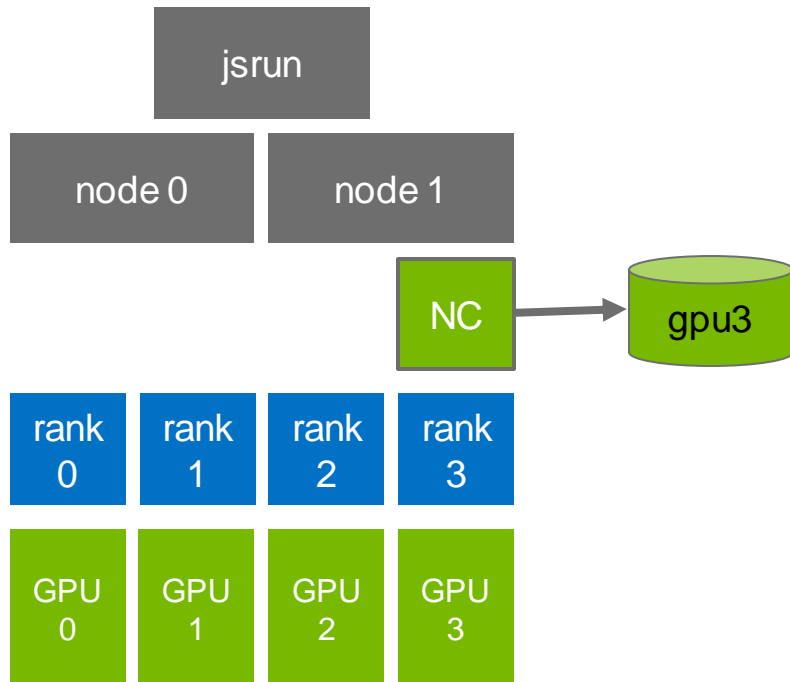
Multi-Process Profiling



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
```