Nsight Compute

41TH VI-HPS TUNING WORKSHOP

Felix Schmitt

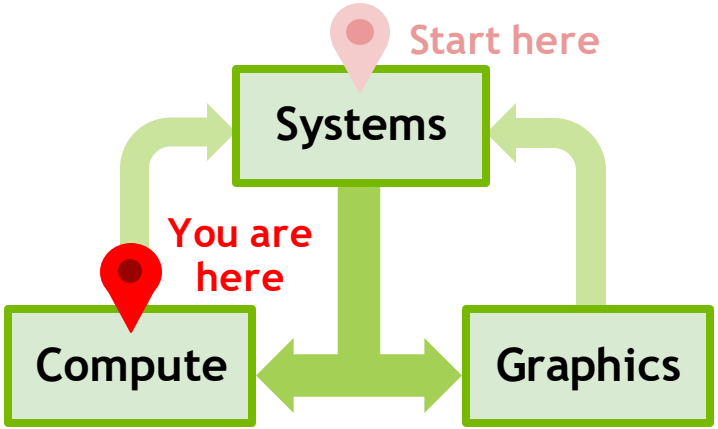
Nsight Product Family

Nsight Systems - Analyze application algorithms system-wide

Nsight Compute - Analyze CUDA kernels

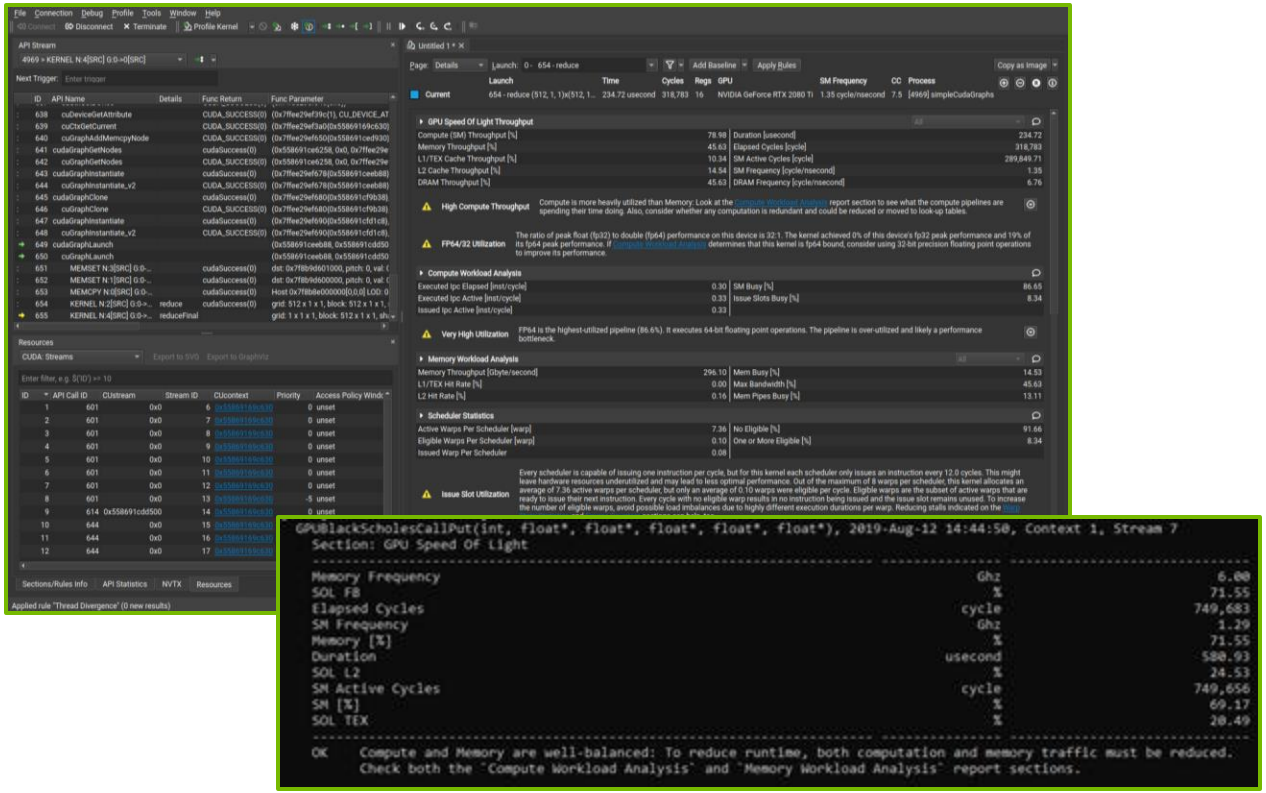
Nsight Graphics - Debug/analyze graphics workloads

Workflow




Overview

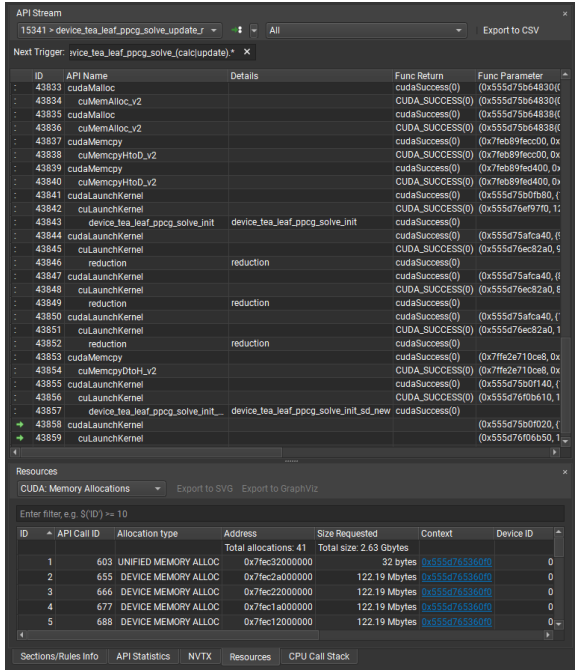
Overview



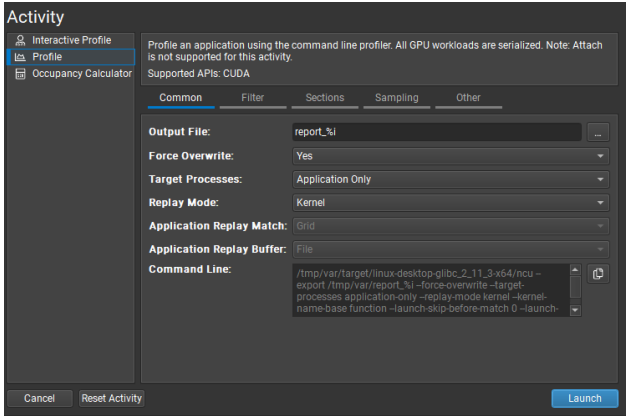
- Interactive CUDA Kernel profiler
- Targeted metric sections for various performance aspects
- Customizable data collection and presentation (tables, charts, …)
- UI and Command Line
- Python-based API for guided analysis and post-processing
- Support for remote profiling across machines and platforms

Profiling Activities

Interactive Profile



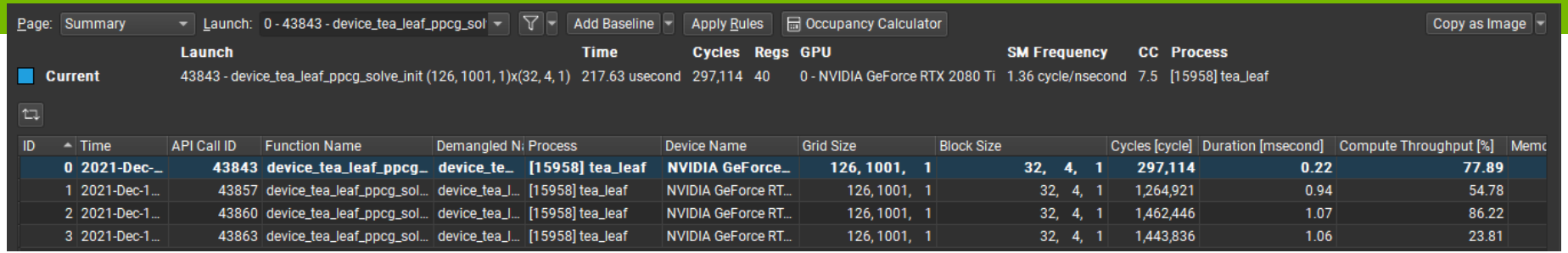
(Non-interactive) Profile

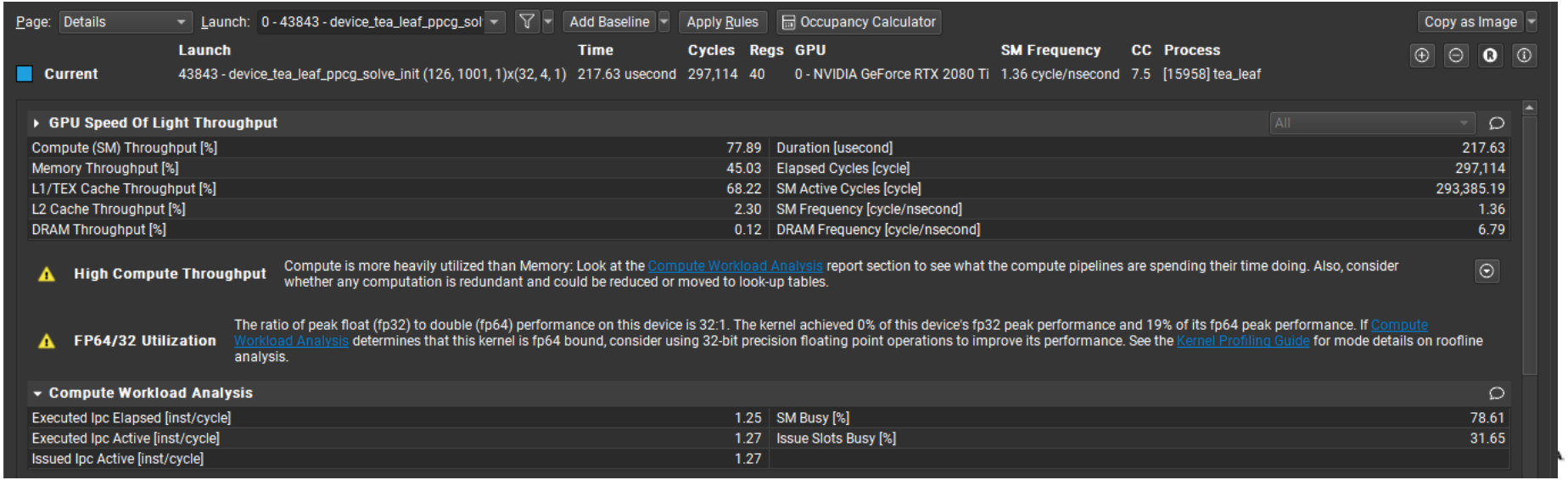


Command Line

```
$~/working/git/TeaLeaf_CUDA$ /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: PPCG
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/nsecond                           6.63
    SM Frequency                                                           cycle/nsecond                           1.33
    Elapsed Cycles                                                                 cycle                      1,457,762
    Memory [%]                                                                         %                          91.70
    DRAM Throughput                                                                    %                          91.70
    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

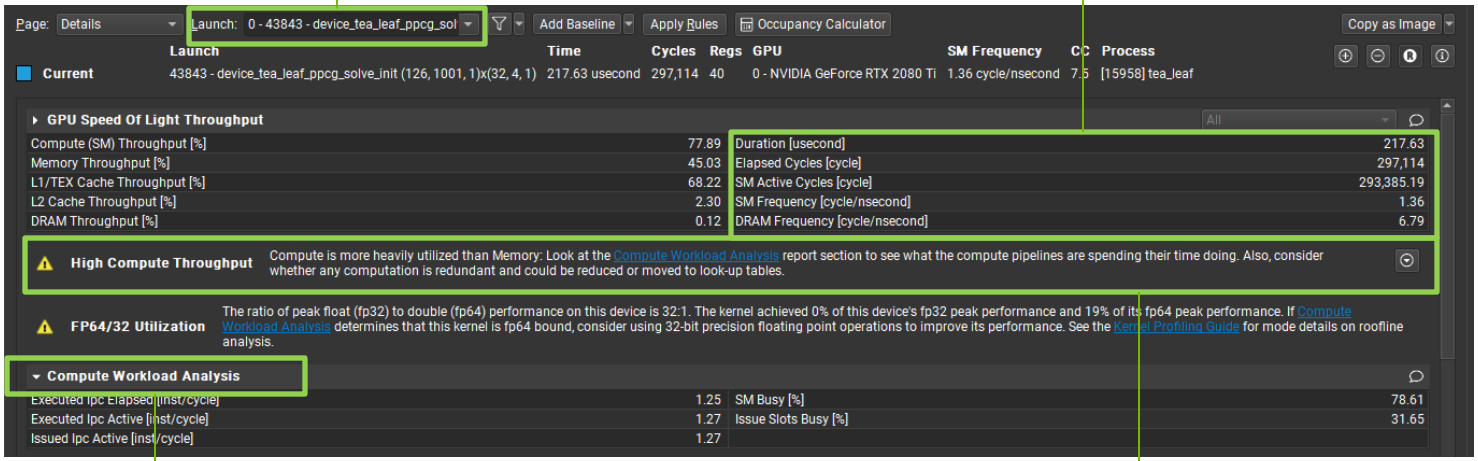




Profiler Report

Selected result

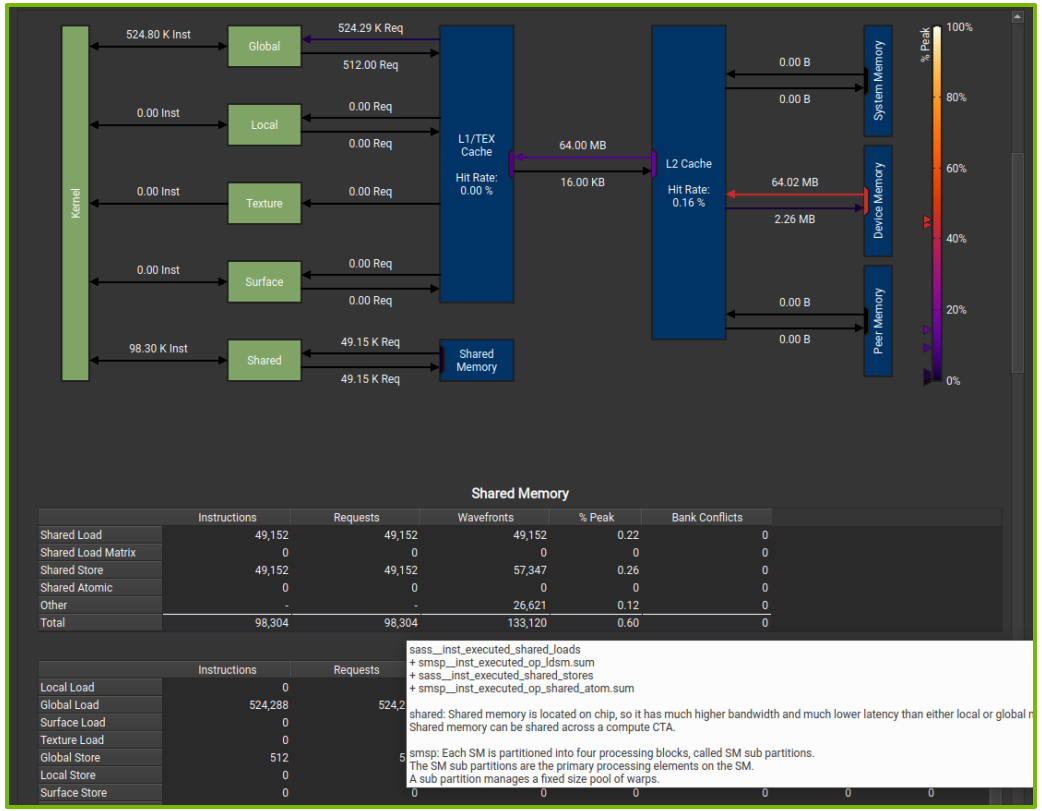
Metric values



Expandable Sections

Expert Analysis (Rules)

Profiler Report

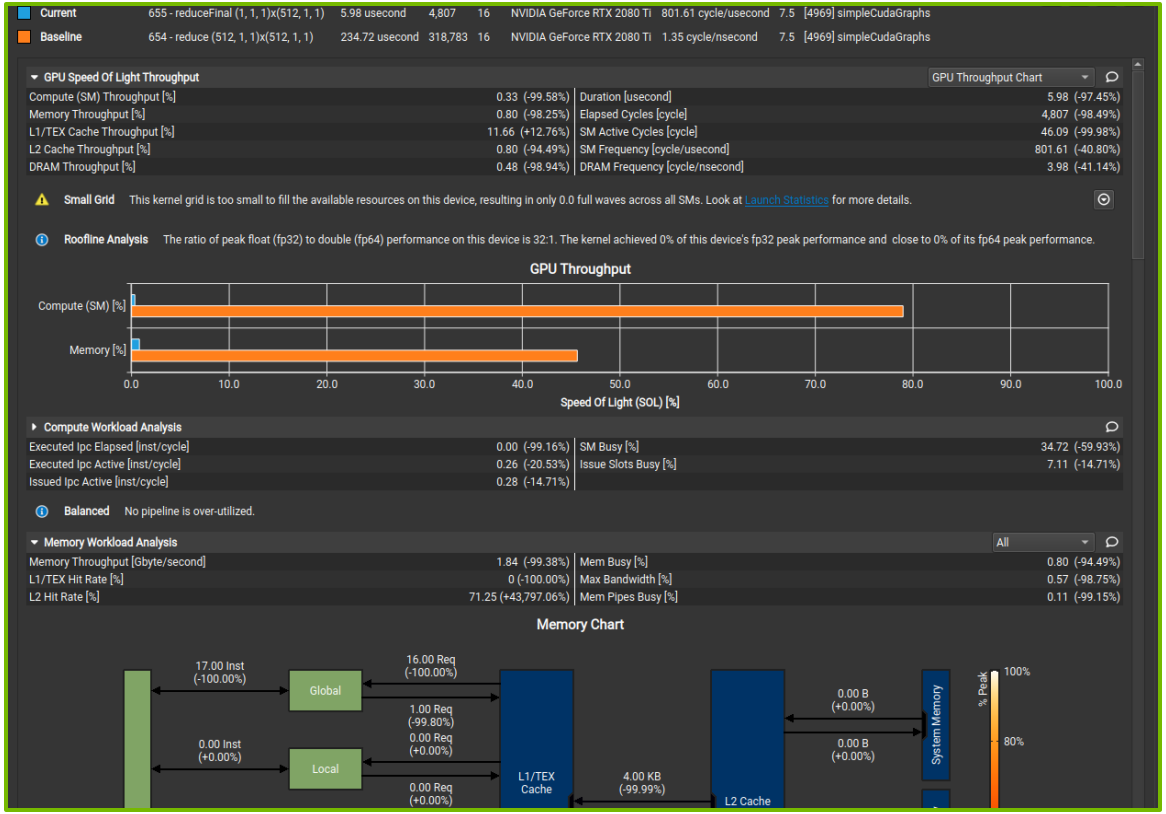


Detailed memory workload analysis chart and tables

Shows transferred data or throughputs

Tooltips provide metric names, calculation formulas and detailed background info

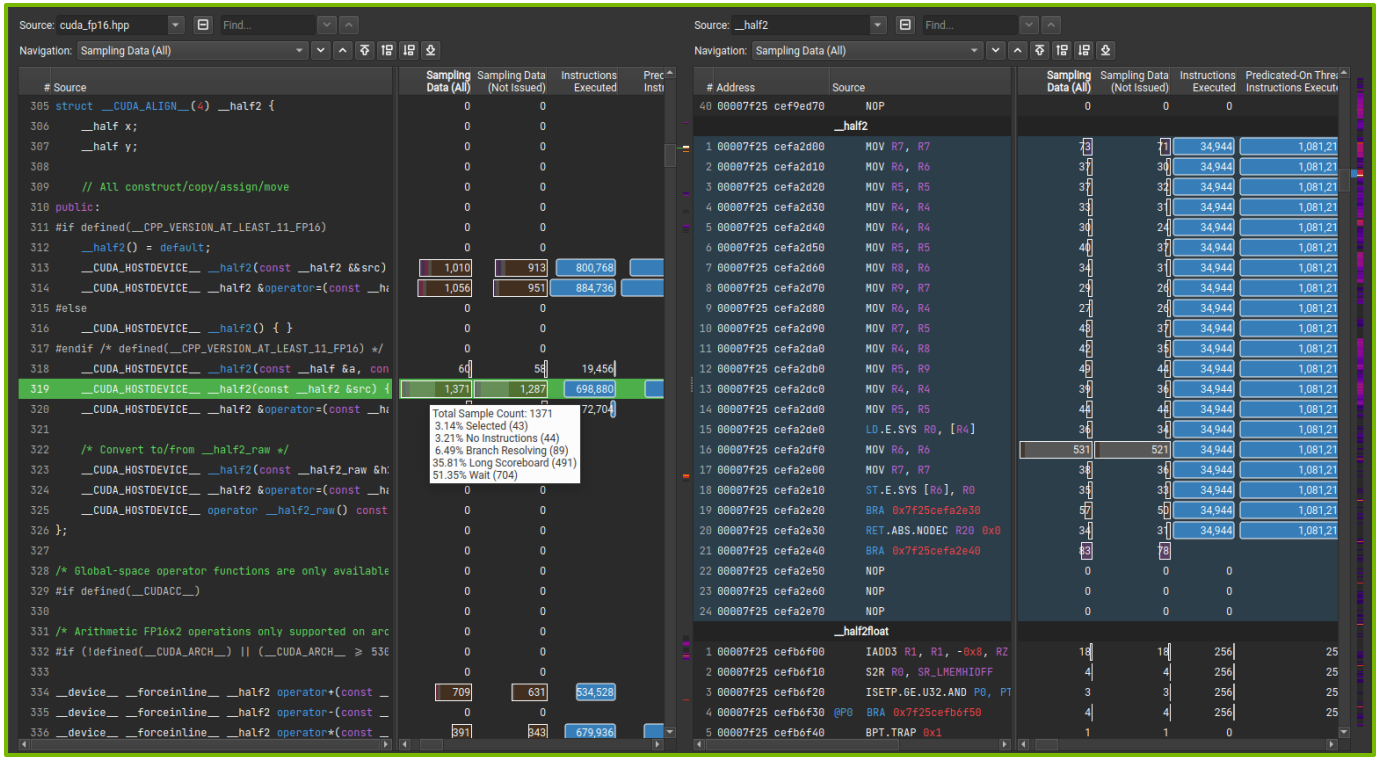
Profiler Report



Comparison of results directly within the tool with "Baselines"

Supported across kernels, reports, and GPU architectures

Profiler Report

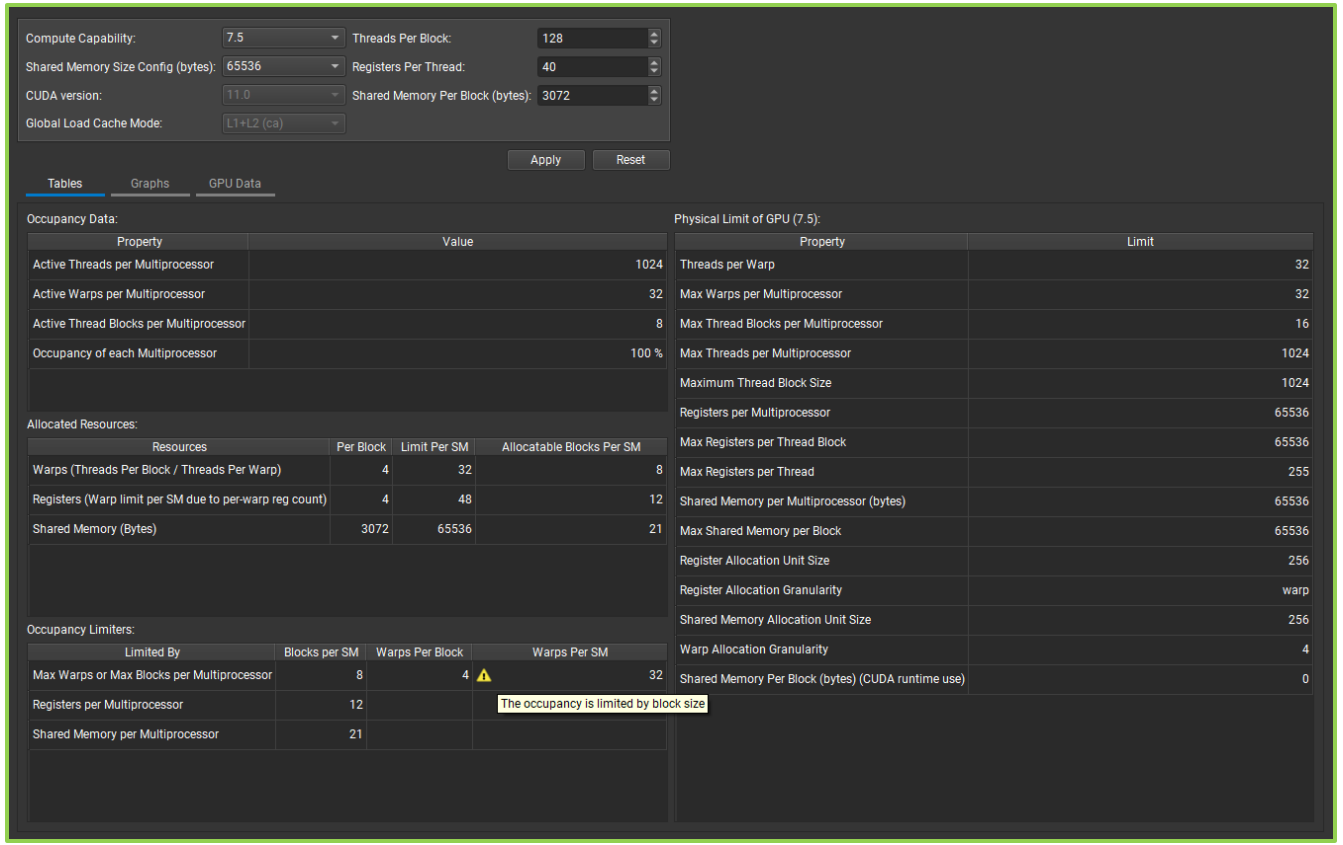


Source/PTX/SASS analysis and correlation

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

Metric heatmap

Occupancy Calculator




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

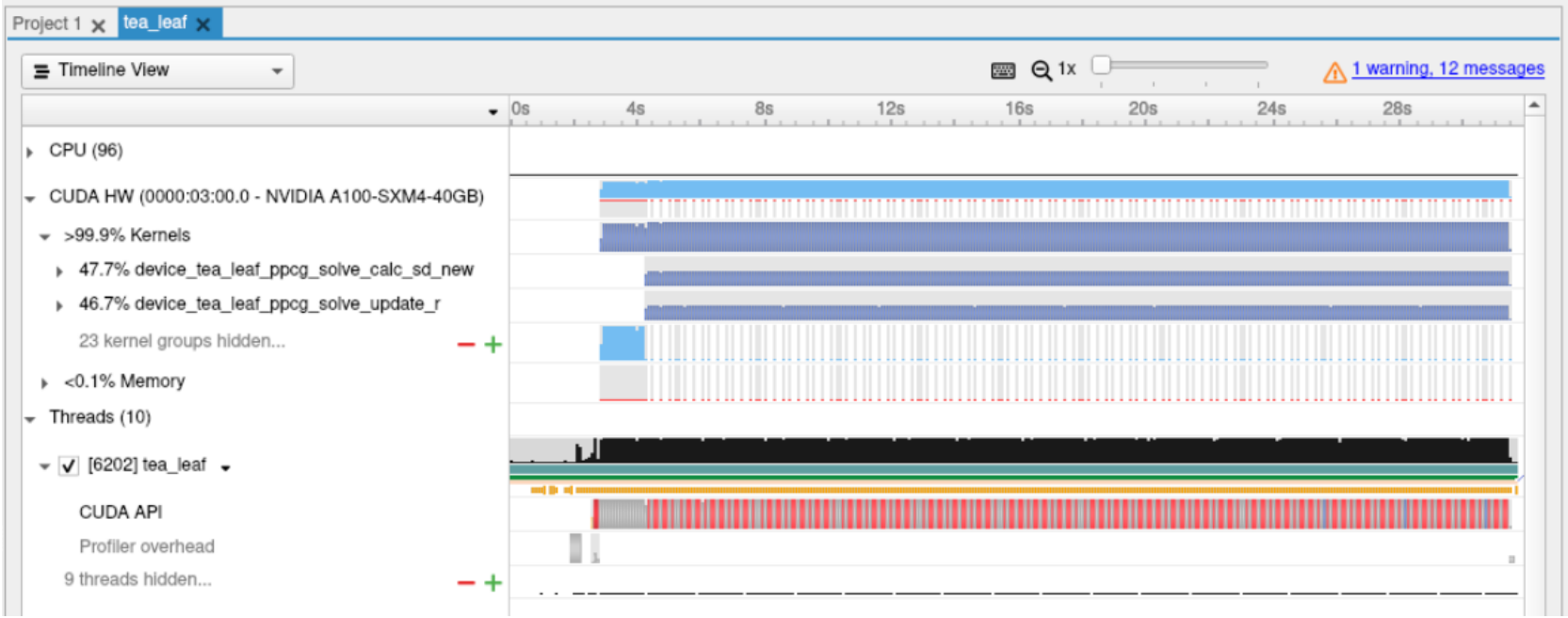
```
$ make
```

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



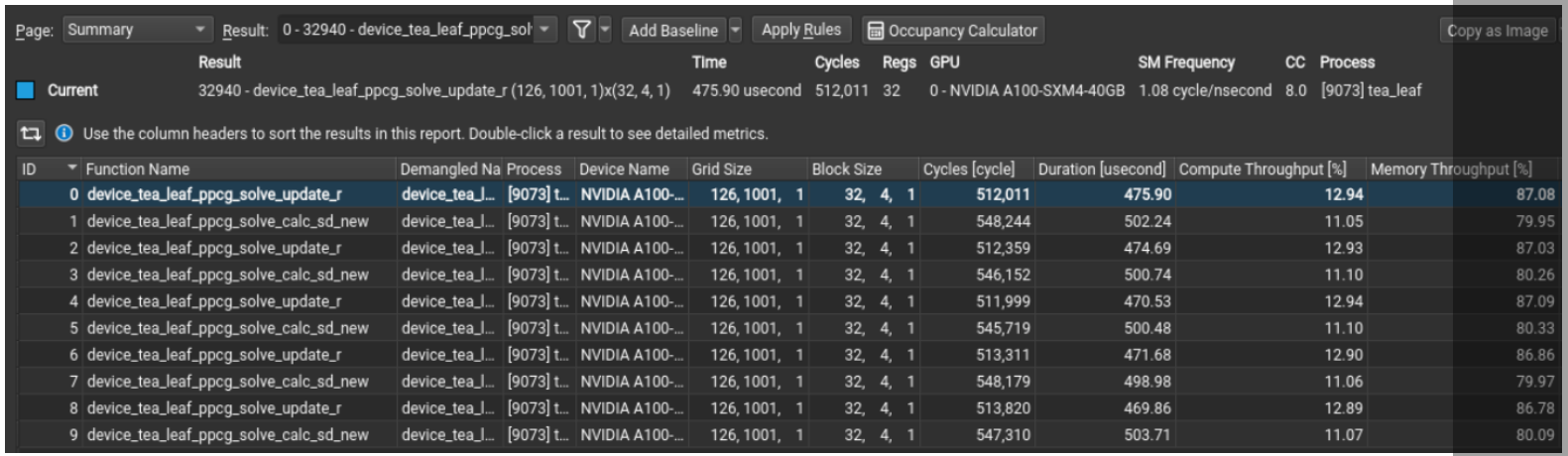
Profiling with Nsight Compute

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
```

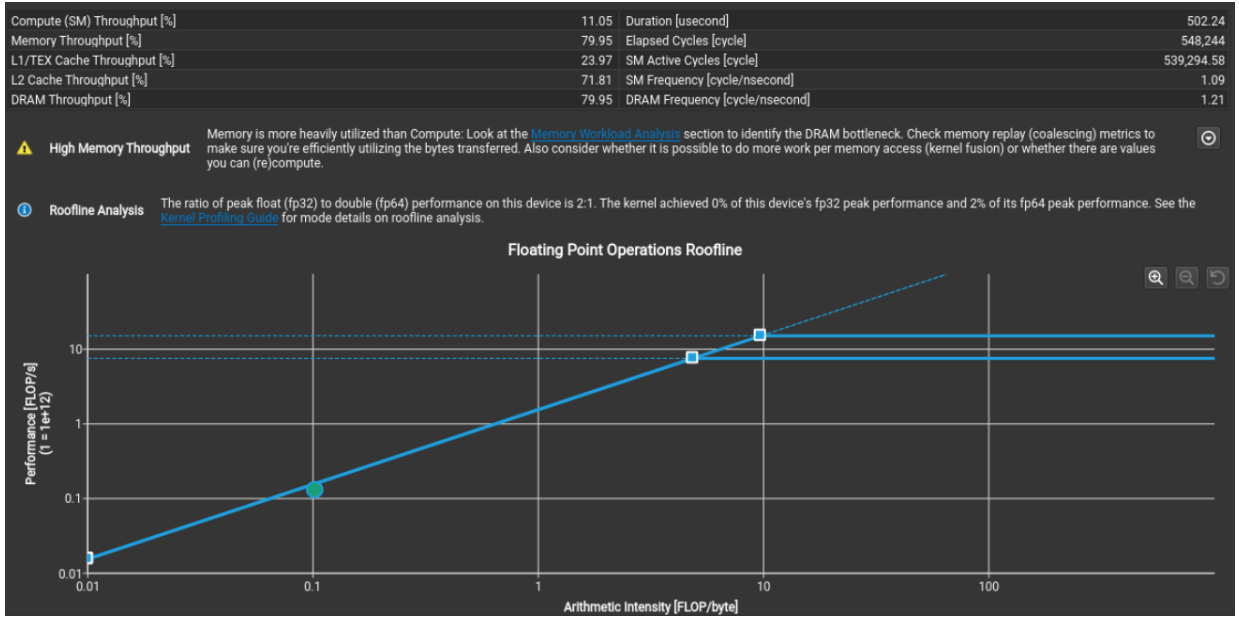
Summary page confirms that all instances of each respective kernel have similar performance characteristics - focus on a single instance for each

Live



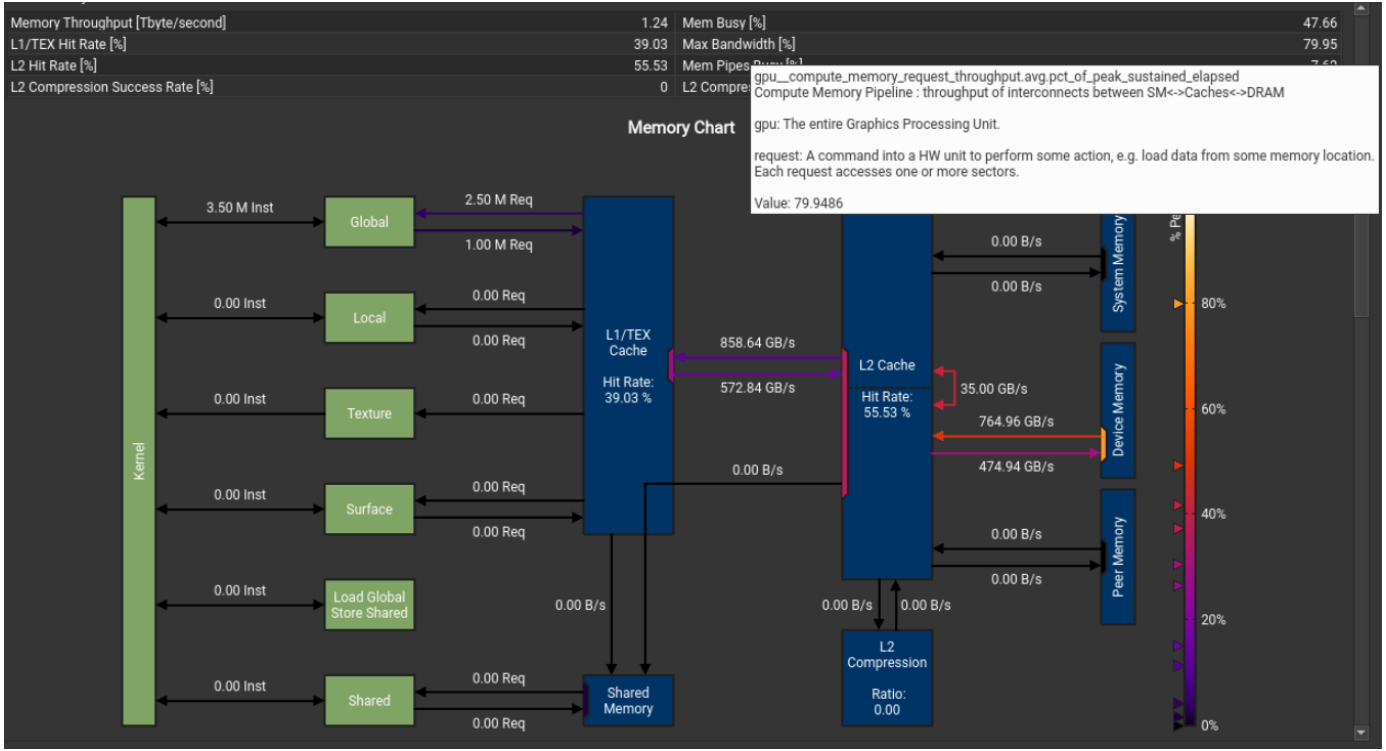
Analyzing calc kernel

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

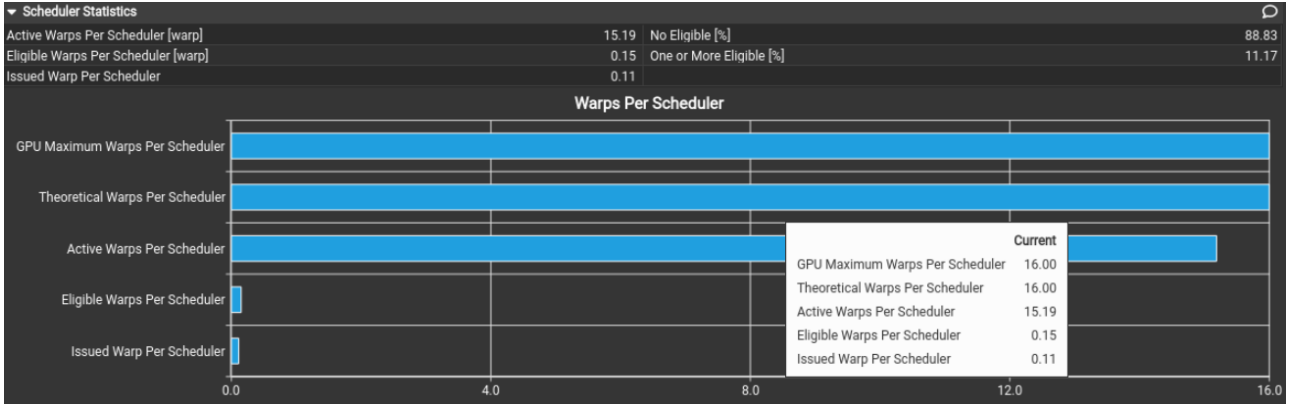


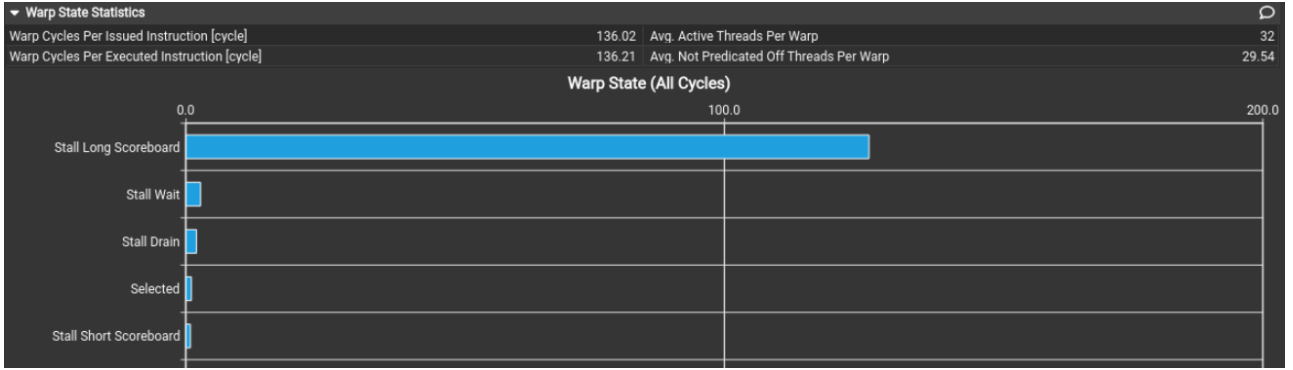
Analyzing calc kernel

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



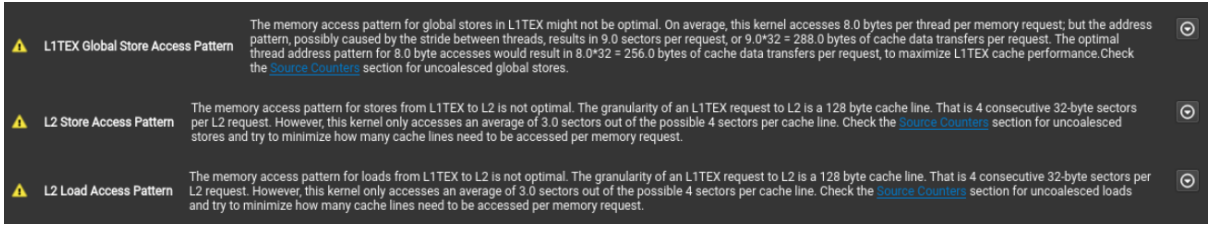


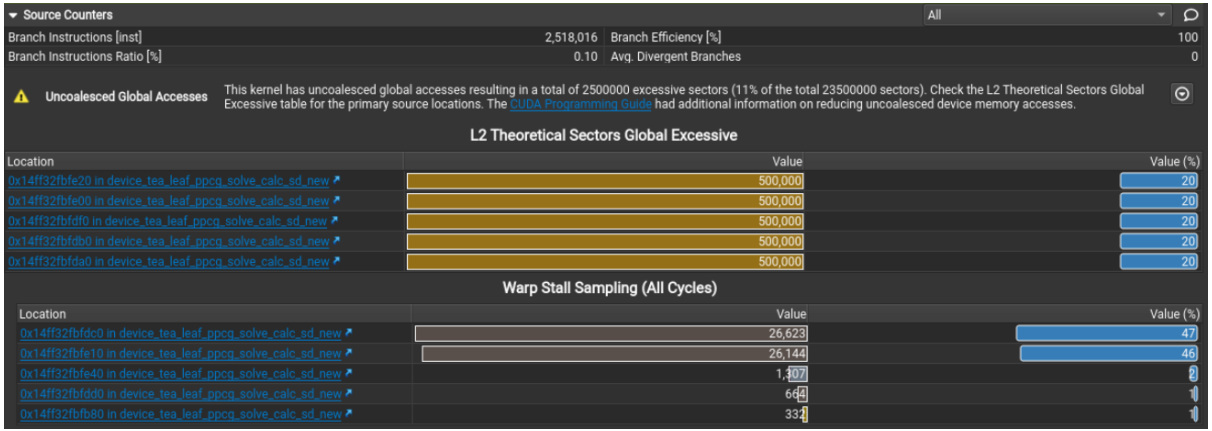
Analyzing calc kernel

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

Source Counters show uncoalesced accesses and location of the stalls

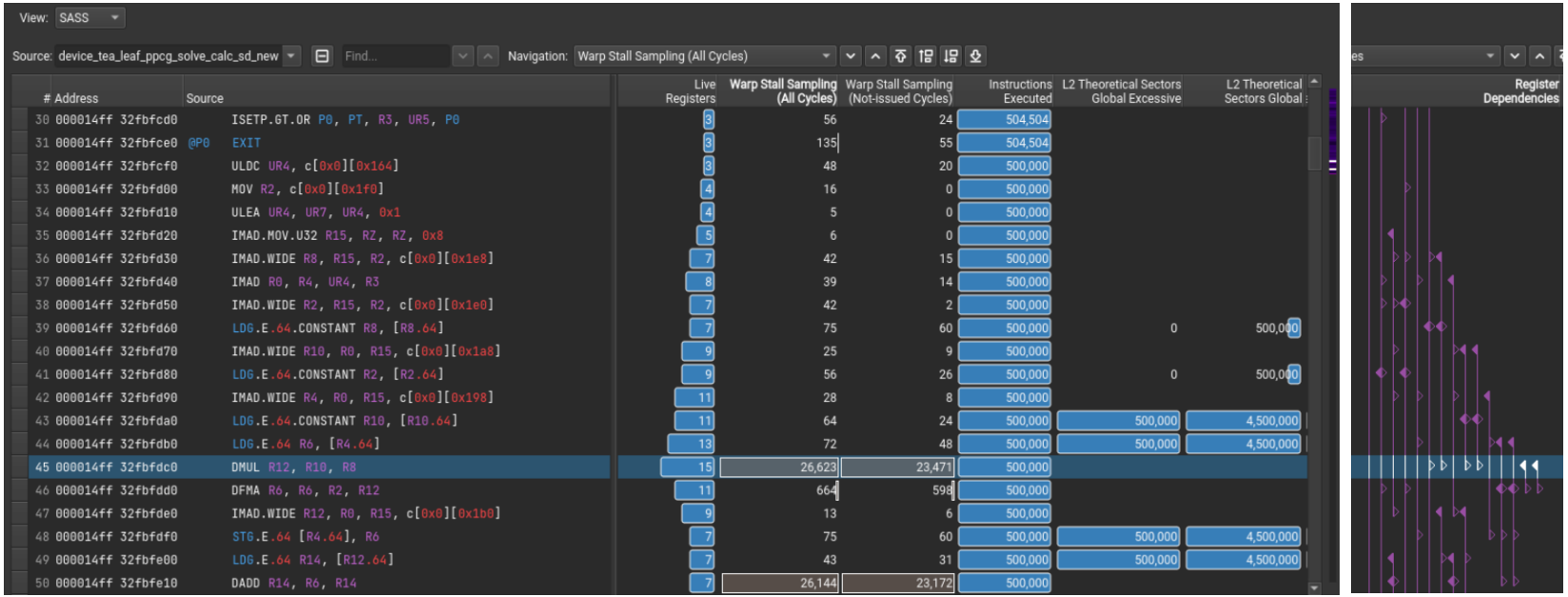
Jump to Source page via this link





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



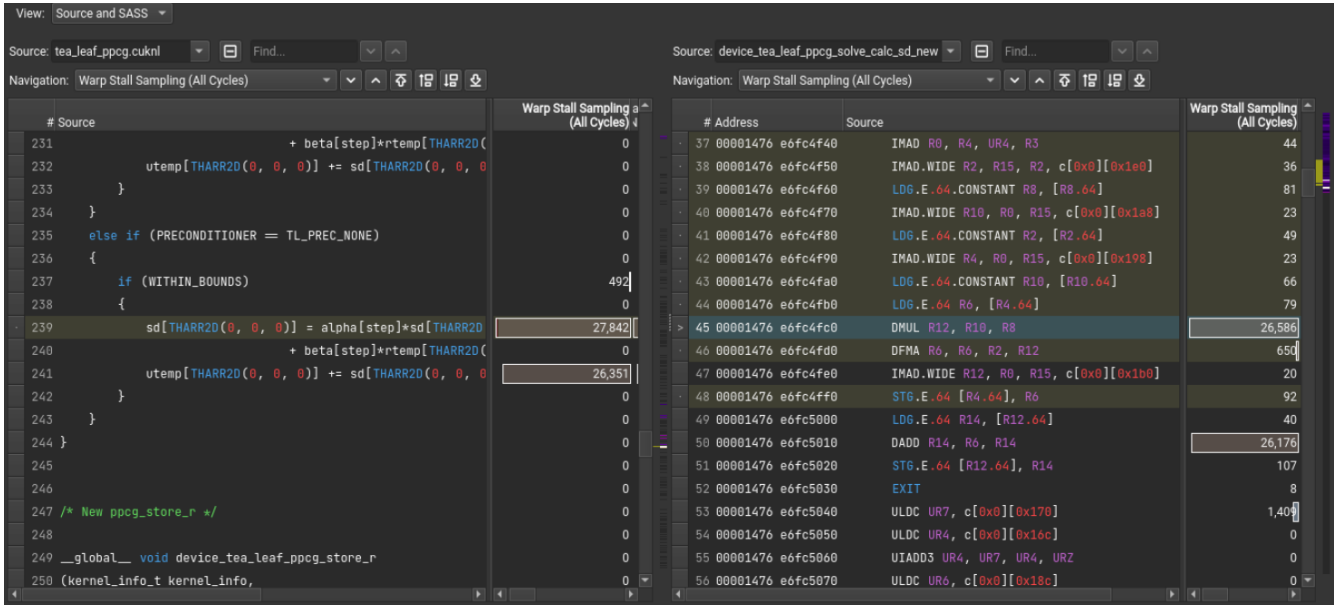
#	Address	Source	Live Registers	Warp Stall Sampling (All Cycles)	Warp Stall Sampling (Not-issued Cycles)	Instructions Executed	L2 Theoretical Sectors Global Excessive	L2 Theoretical Sectors Global
30	000014ff 32fbfcd0	ISETP.GT.OR P0, PT, R3, UR5, P0	3	56	24	504,504		
31	000014ff 32fbfce0	@P0 EXIT	3	135	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, RZ, RZ, 0x8	5	6	0	500,000		
36	000014ff 32fbfd30	IMAD.WIDE R8, R15, R2, c[0x0][0x1e8]	7	42	15	500,000		
37	000014ff 32fbfd40	IMAD R0, R4, UR4, R3	8	39	14	500,000		
38	000014ff 32fbfd50	IMAD.WIDE R2, R15, R2, 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][0x198]	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	664	598	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 (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



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
```


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, MemoryWorkloadAnaly 157
           sis, Occupancy, SchedulerStats, SourceCounters, SpeedOfLight, SpeedOfLight_
           RooflineChart, WarpStateStats
full       ComputeWorkloadAnalysis, InstructionStats, LaunchStats, MemoryWorkloadAnaly 162
           sis, MemoryWorkloadAnalysis_Chart, MemoryWorkloadAnalysis_Tables, Occupancy
           , SchedulerStats, SourceCounters, SpeedOfLight, SpeedOfLight_RooflineChart,
            WarpStateStats
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 ga100
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

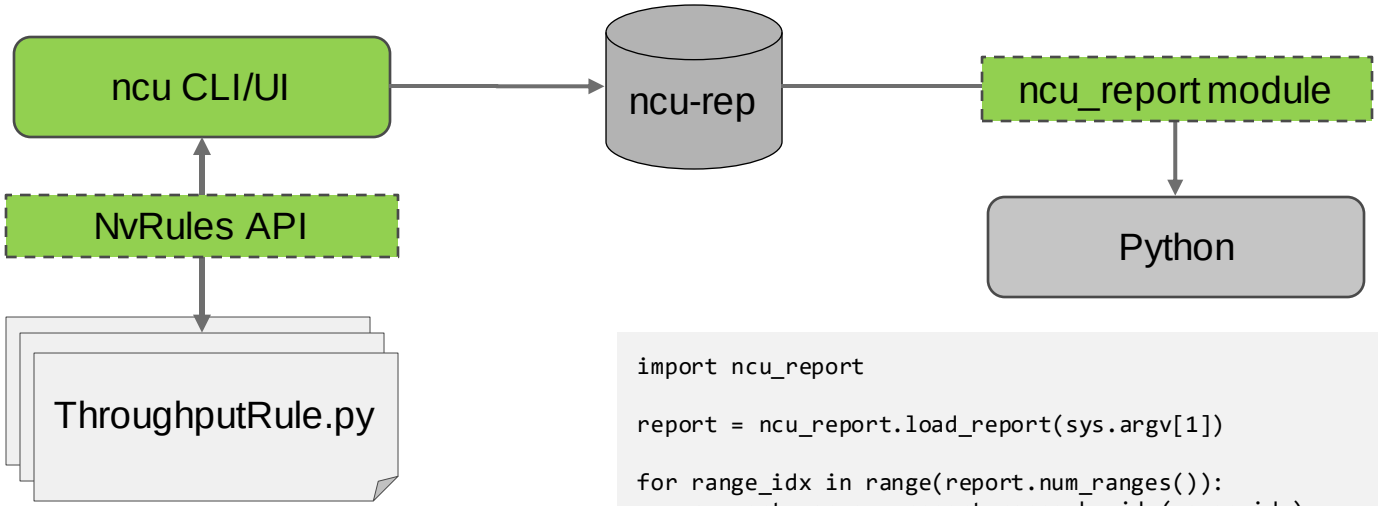
SASS (assembly) is always available, embedded into the report
CUDA-C (Source) and PTX availability depends on compilation flags
Use -lineinfo to include source/SASS correlation data in the binary

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


More: Data Analysis

Python Interfaces

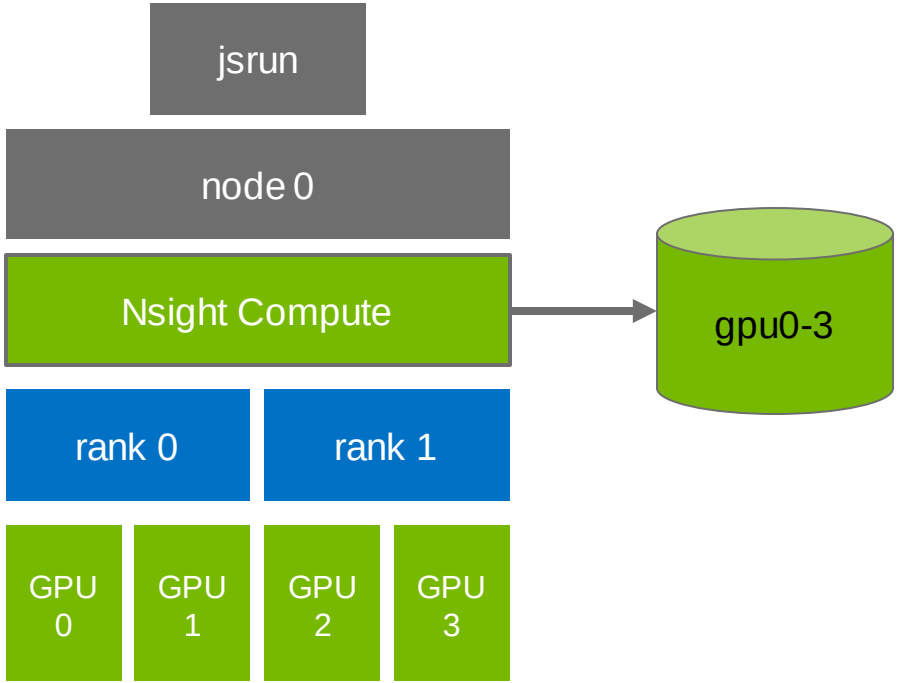



```
import ncu_report

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

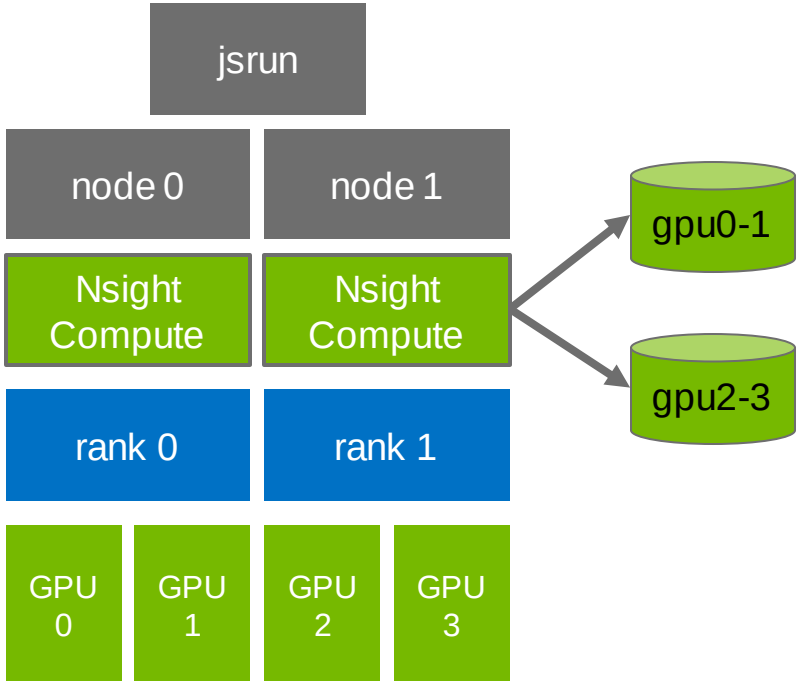


On a single-node submission, one Nsight Compute instance can profile all launched child processes

Data for all processes is stored in one report file

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

Multi-Process Profiling

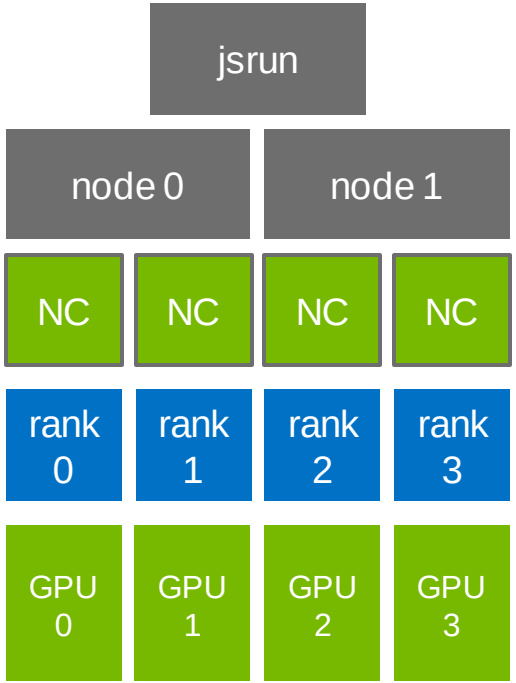


On multi-node submissions, one tool instance can be used per node

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>
```

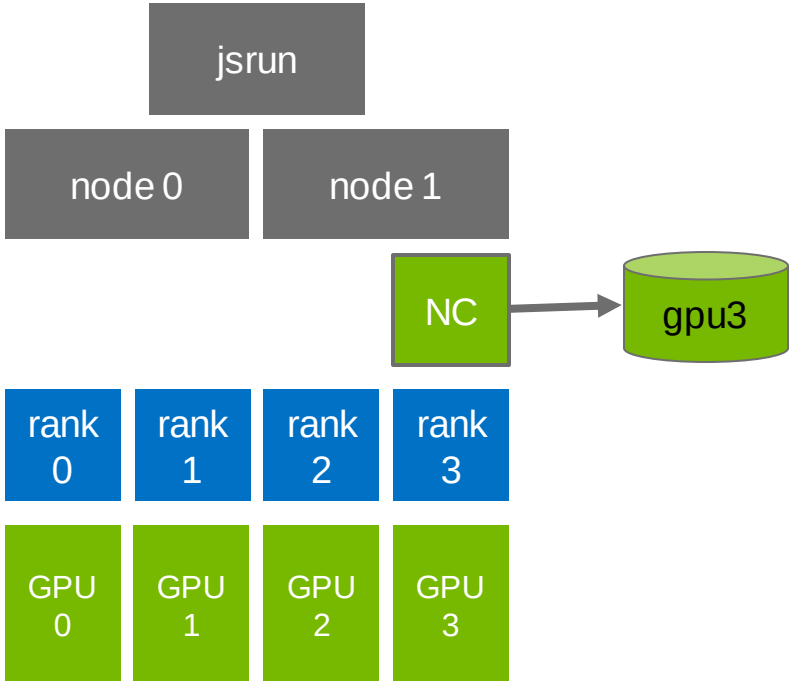
Multi-Process Profiling



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