Nsight Compute

OLCF Webinar

Felix Schmitt, Mahen Doshi, Jonathan Vincent

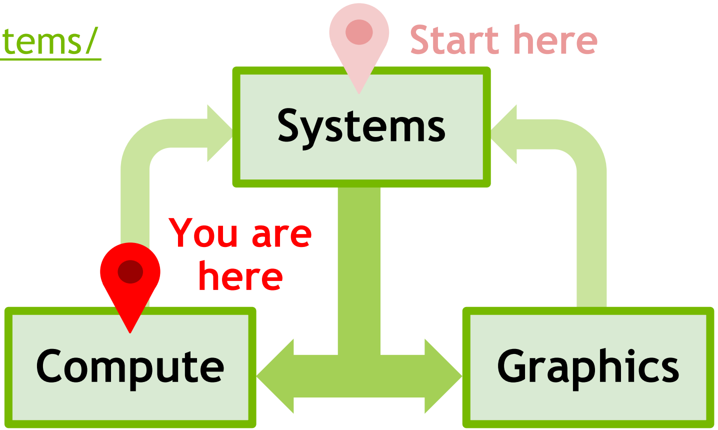
Nsight Product Family

Nsight Systems - Analyze application algorithms system-wide
https://www.olcf.ornl.gov/calendar/nvidia-profiling-tools-nsight-systems/

Nsight Compute - Analyze CUDA kernels

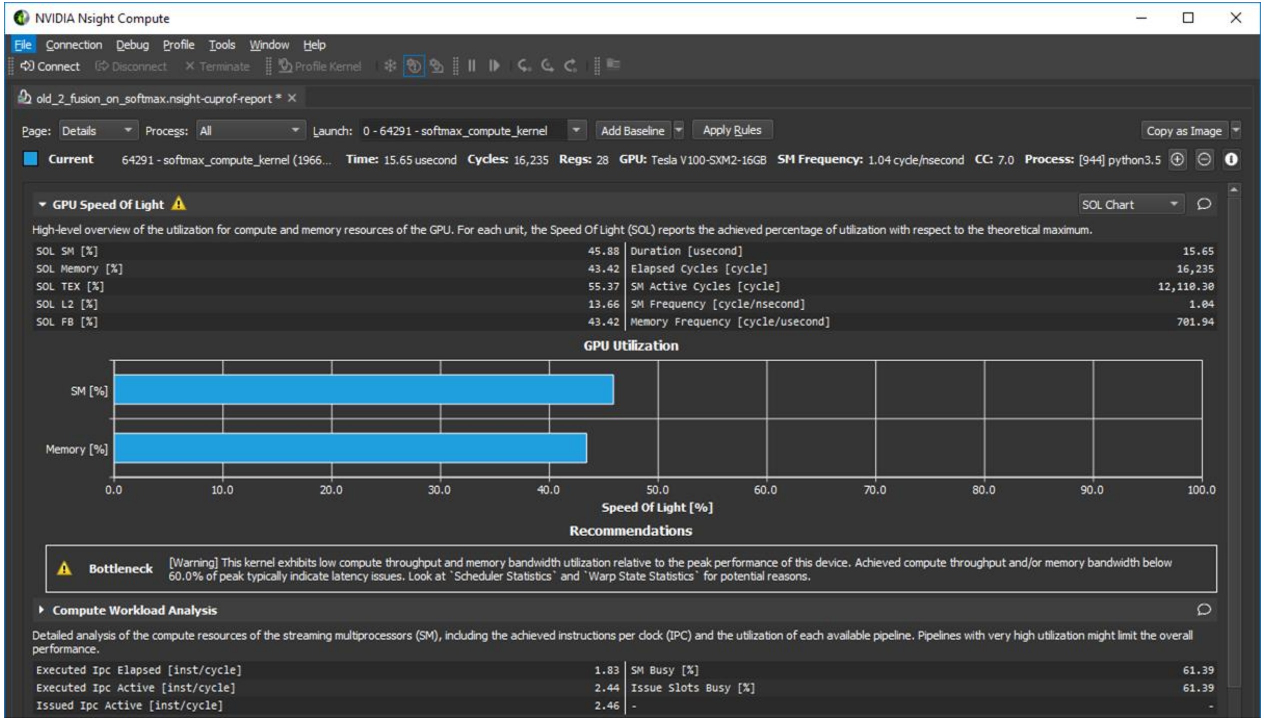
Nsight Graphics - Debug/analyze graphics workloads

Workflow



Nsight Compute

Nsight Compute



CUDA Kernel profiler

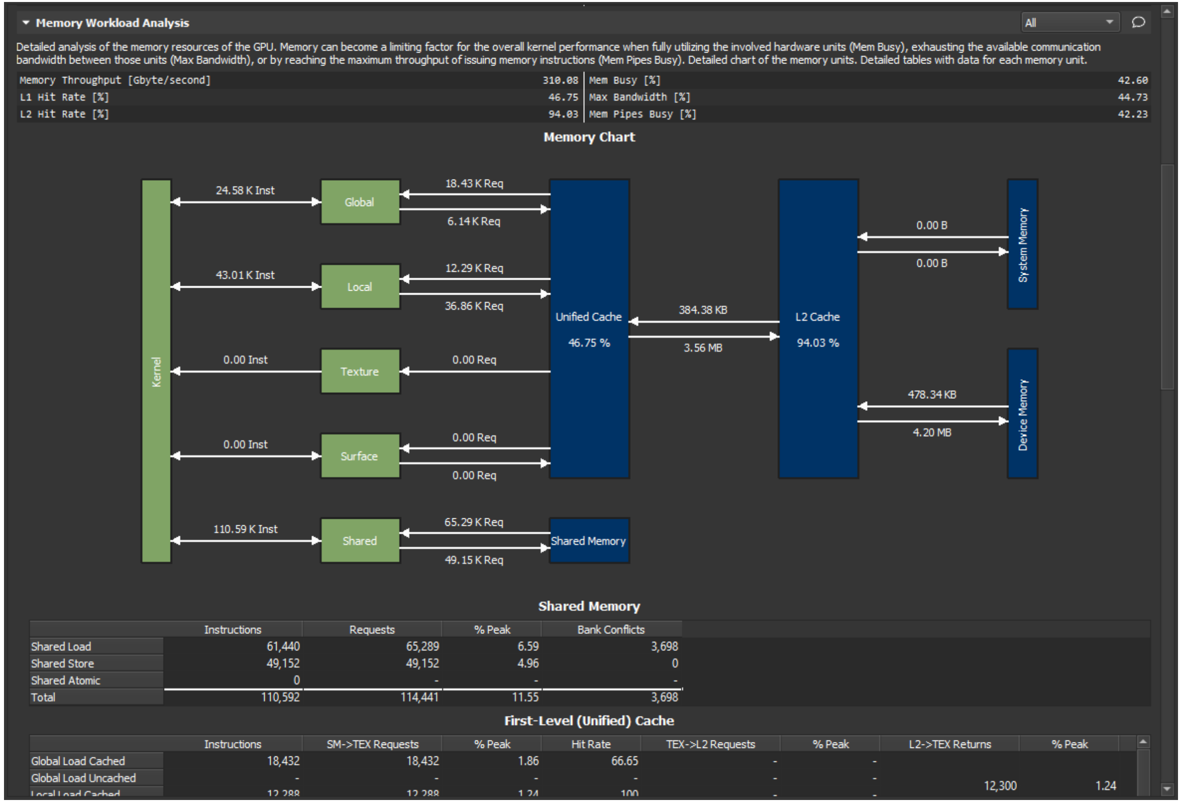
Targeted metric sections for various performance aspects

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

UI and Command Line

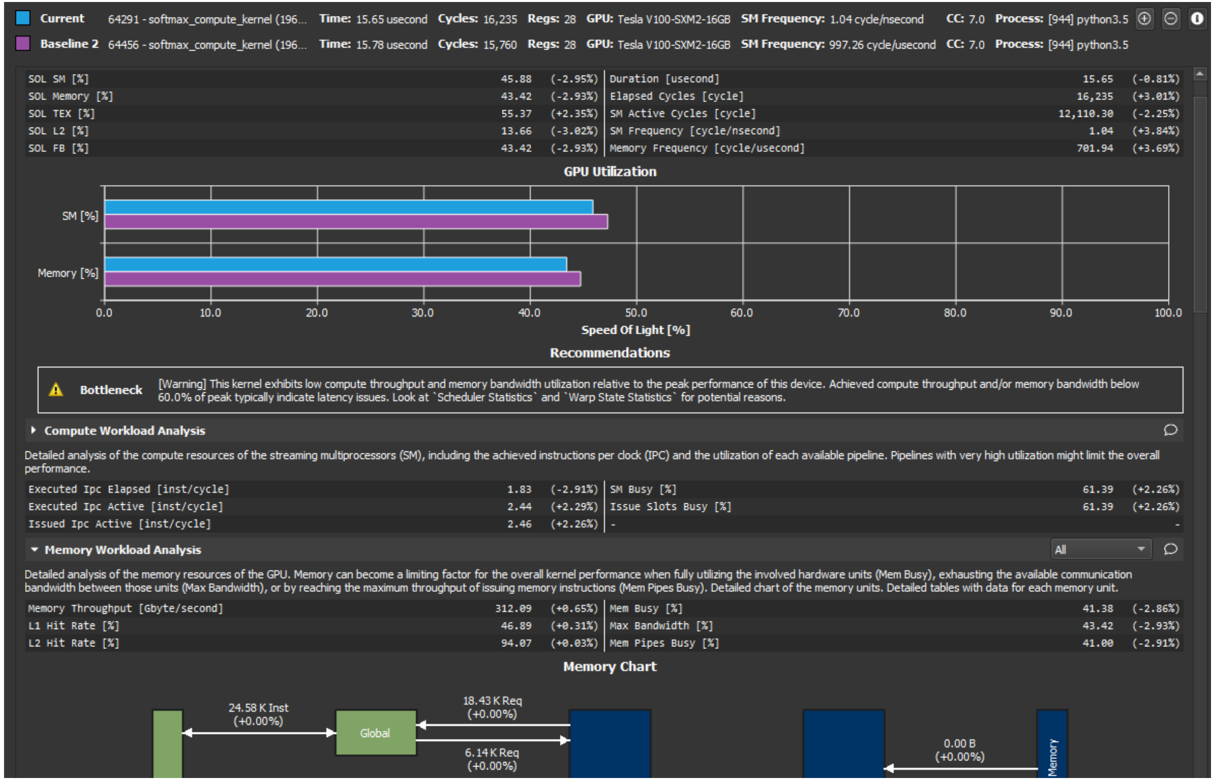
Python-based rules for guided analysis (or post-processing)

Nsight Compute



Detailed memory workload analysis chart and tables

Nsight Compute

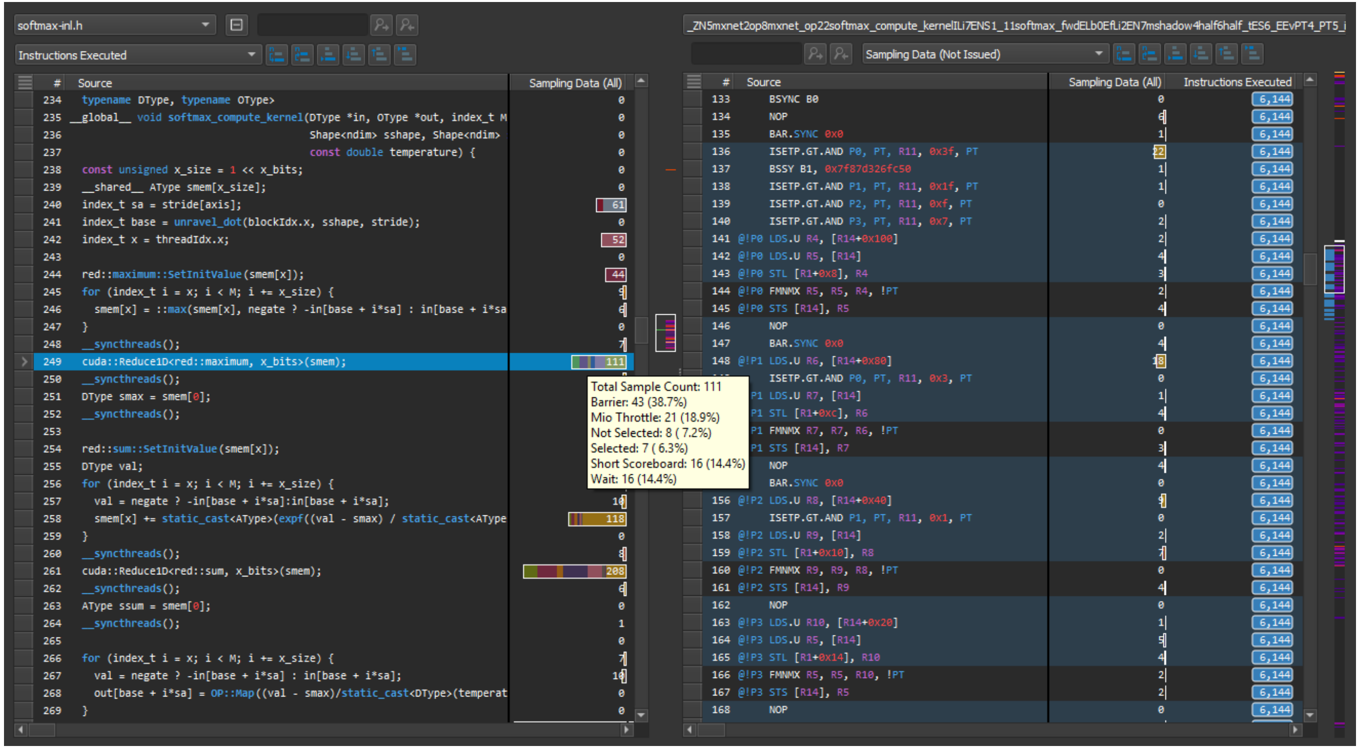


Comparison of results directly within the tool with "Baselines"

Supported across kernels, reports, and GPU architectures

NVIDIA

Nsight Compute



Source/PTX/SASS analysis and correlation

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

Metric heatmap

Nsight Compute

```
==PROF== Disconnected from process 8792
[8792] CuBlackscholes.exe@127.0.0.1
  GPUBlackScholesCallPut(int, float*, float*, float*, float*, float*), Block Size 256,
    Section: GPU Speed Of Light
    Metric Name                 Metric Unit Minimum       Maximum       Average
    --------------------------- ----------- ------------- ------------- -------------
    dram__frequency             Ghz         6.004059      6.004059      6.004059
    fbpa__sol_pct               %           70.191350     70.191350     70.191350
    gpc__elapsed_cycles_max     cycle       751140.000000 751140.000000 751140.000000
    gpc__frequency              Ghz         1.287365      1.287365      1.287365
    gpu__compute_memory_sol_pct %           70.191350     70.191350     70.191350
    gpu__time_duration          usecond     583.456000    583.456000    583.456000
    ltc__sol_pct                %           24.488190     24.488190     24.488190
    sm__elapsed_cycles_avg      cycle       751121.000000 751121.000000 751121.000000
    sm__sol_pct                 %           69.036830     69.036830     69.036830
    tex__sol_pct                %           20.449435     20.449435     20.449435
```

```
  GPUBlackScholesCallPut(int, float*, float*, float*, float*, float*), 2019-Aug-12 14:44:50, Context 1, Stream 7
    Section: GPU Speed Of Light
    ---------------------------------------------------------------------- --------------- ------------------------------
    Memory Frequency                                                                   Ghz                           6.00
    SOL FB                                                                               %                          71.55
    Elapsed Cycles                                                                   cycle                        749,683
    SM Frequency                                                                       Ghz                           1.29
    Memory [%]                                                                           %                          71.55
    Duration                                                                       usecond                         580.93
    SOL L2                                                                               %                          24.53
    SM Active Cycles                                                                 cycle                        749,656
    SM [%]                                                                               %                          69.17
    SOL TEX                                                                              %                          20.49
    ---------------------------------------------------------------------- --------------- ------------------------------
    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.
```

Full command line interface (CLI) for data collection and analysis

On your workstation

Support for remote profiling across machines, platforms (Linux, Windows, …) in UI and CLI

Nsight Compute on Summit

Loading Module

Use nv-nsight-cu-cli command line interface for data collection in batch environments

Available as part of the CUDA toolkit

```
$ module load cuda/10.1.243
$ /sw/summit/cuda/10.1.243/nsight-compute/nv-nsight-cu-cli
```

Or as standalone installation (e.g. newer release than CUDA)

```
$ module load nsight-compute/2019.5.0
$ /sw/summit/nsight-compute/2019.5.0/nv-nsight-cu-cli
```

Collecting Data

By default, results are printed to stdout

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

```
$ nv-nsight-cu-cli -f -o $HOME/my_report <app>
$ my_report.nsight-cuprof-report
```

Use (env) vars available in your batch script to add report name placeholders

```
$ nv-nsight-cu-cli -f -o $HOME/my_report_${LSB_JOBID} <app>
$ my_report_951697.nsight-cuprof-report
```

Full parity with nvprof filename placeholders/file macros in next tool version

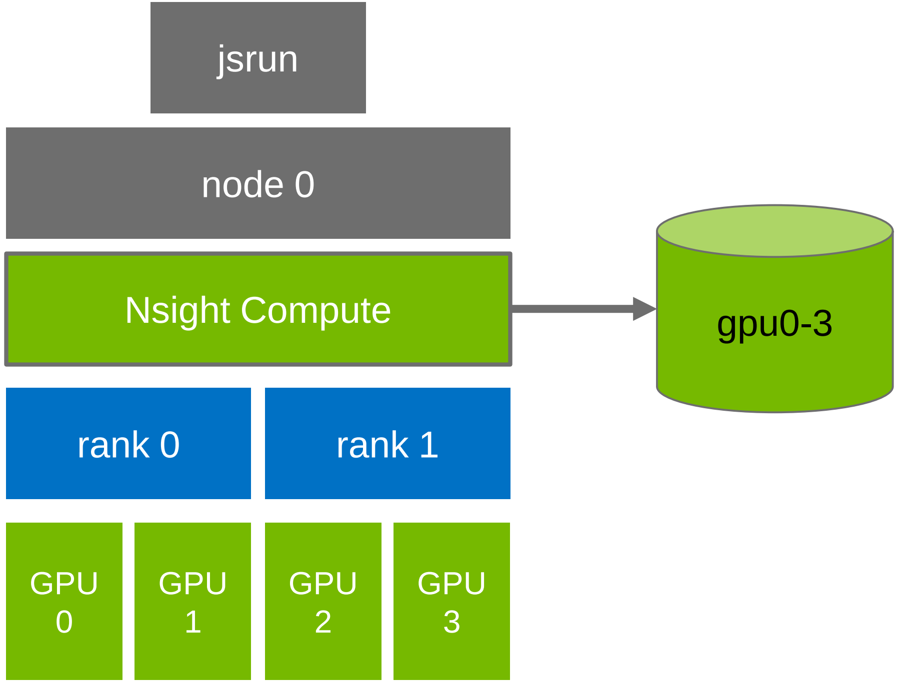
Disabling PAMI hooks for Spectrum MPI might be required, depending on your application

```
$ jsrun … --smpiargs "-disable_gpu_hooks” …
```

This can be an issue if your application requires

```
$ jsrun … --smpiargs "-gpu” …
```

Multi-Process Profiling

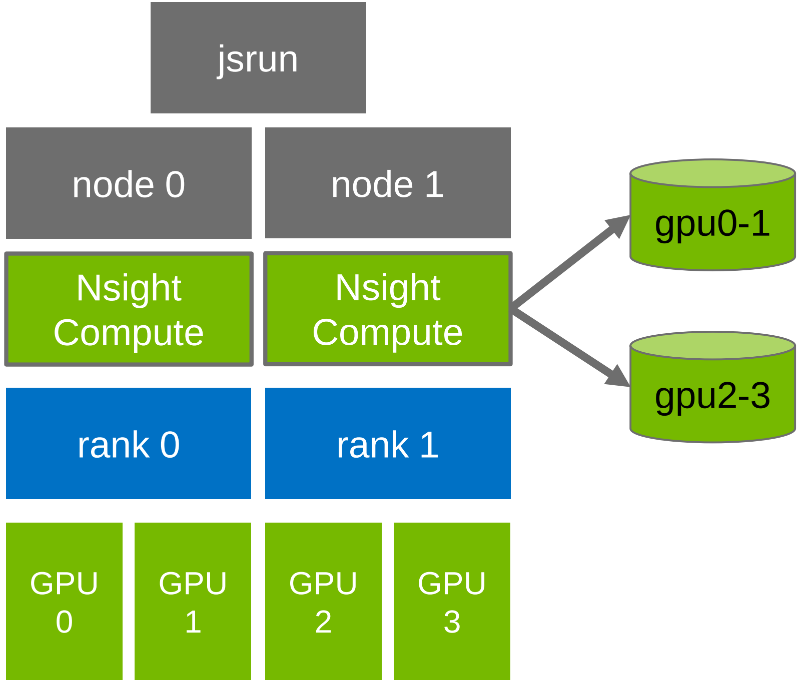


On a single-node submission, Nsight Compute can profile all launched processes

Data for all processes is stored in one report file

```
nv-nsight-cu-cli --target-processes all
-o <single-report-name> <app> <args>
```

Multi-Process Profiling

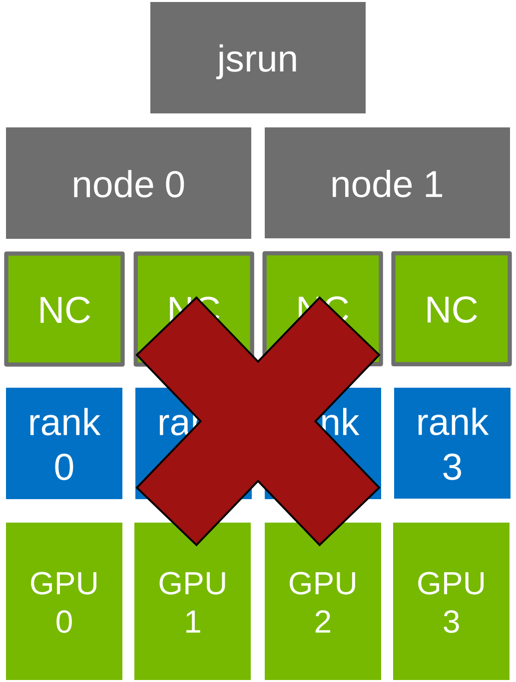


On multi-node submissions, at most one tool instance can be used per node

Ensure that instances don't write to the same report file

```
nv-nsight-cu-cli -o
report_$OMPI_COMM_WORLD_RANK <app>
<args>
```

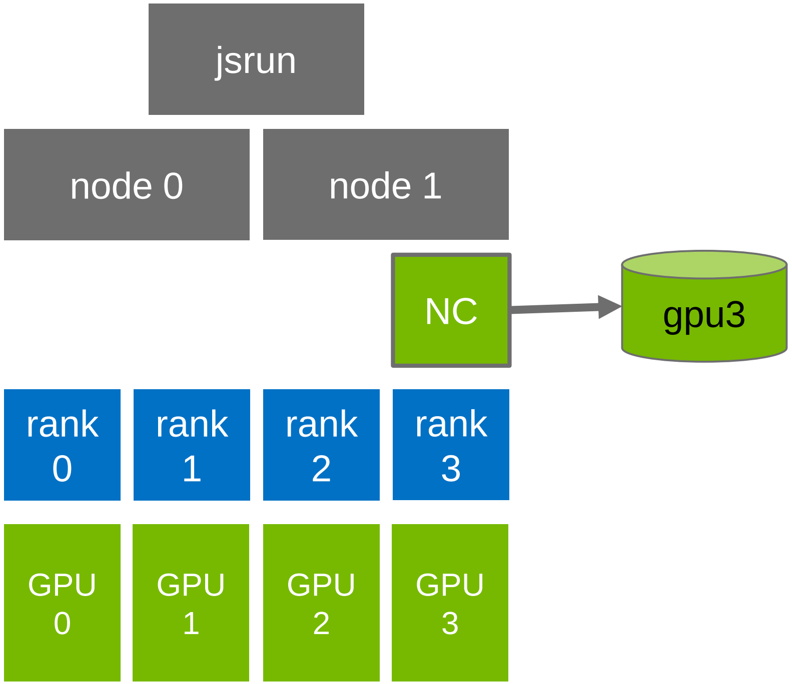
Multi-Process Profiling



Multiple tool instances on the same node are currently not supported

This will be fixed in the next version

Multi-Process Profiling



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

```
#!/bin/bash
if [[ "$OMPI_COMM_WORLD_RANK" == "3" ]] ; then
    /sw/summit/cuda/10.1.243/ nsight-
compute/nv-nsight-cu-cli -o
report_${OMPI_COMM_WORLD_RANK}  --target-
processes all $*
else
    $*
fi
```

Retrieving Data

Use the Nsight Compute CLI (nv-nsight-cu-cli) on any node to import and analyze the report (--import)

More common, transfer the report to your local workstation
Reports compress very well, consider tar -czvf before transfer

Reports can be analyzed on any host system (Linux, Windows, Mac) using the local CLI or UI

Analysis in UI is more comprehensive and user-friendly
Analysis in CLI is more easily automated (--csv)

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

```
cmake/gmxManageNvccConfig.cmake:201
macro(GMX_SET_CUDA_NVCC_FLAGS)
     set(CUDA_NVCC_FLAGS "${GMX_CUDA_NVCC_FLAGS};${CUDA_NVCC_FLAGS};-lineinfo")
endmacro()
```

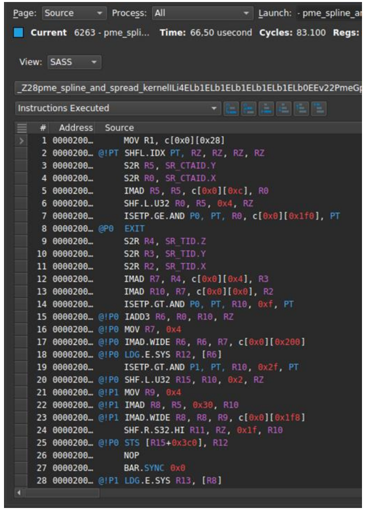
Source is not embedded in the report, need local access to the source file to resolve in the UI

Comparing different iterations (e.g. optimizations) of the same source file can be difficult
Improved in next version to respect file properties

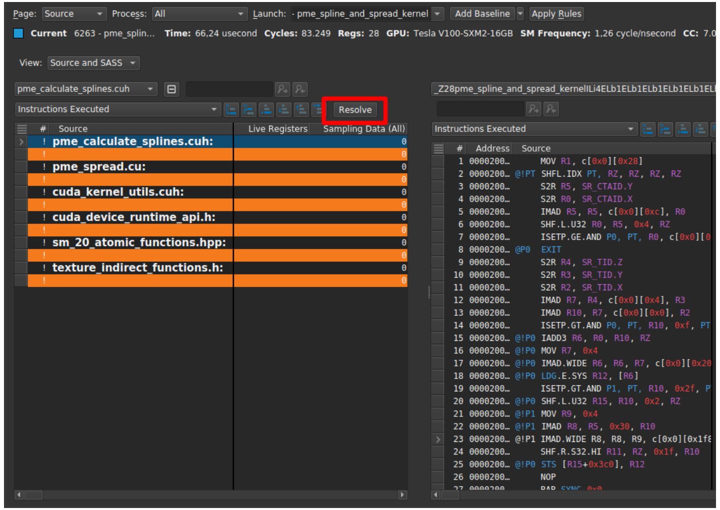
Compiler optimizations can prevent exact source/SASS correlation

Source Analysis

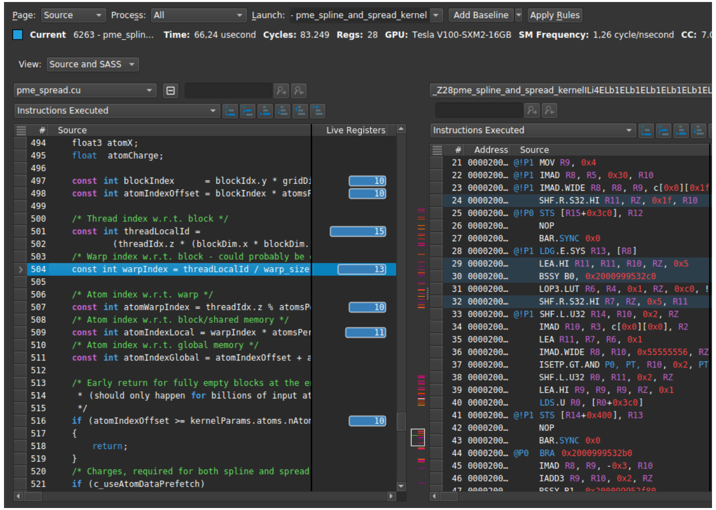
No -lineinfo



-lineinfo, unresolved



-lineinfo, resolved



Transitioning from nvprof to Nsight Compute

nvprof Transition

Check the nvprof (and nvvp) transition guides in the documentation and our blogs
https://docs.nvidia.com/nsight-compute/NsightComputeCli/index.html#nvprof-guide
https://docs.nvidia.com/nsight-compute/NsightCompute/index.html#nvvp-guide
https://devblogs.nvidia.com/migrating-nvidia-nsight-tools-nvvp-nvprof/

Differences	Missing Features (in progress)
New metric names and many more metrics https://docs.nvidia.com/nsight-compute/NsightComputeCli/index.html#nvprof-metric-comparison Cache flush and clock control enabled by default for deterministic data collection Customizable	Application replay No NVLink metrics No trace - use Nsight Systems No MPS support

GROMACS 2020 pme spread/gather Old Version

spline_and_spread: Old Version

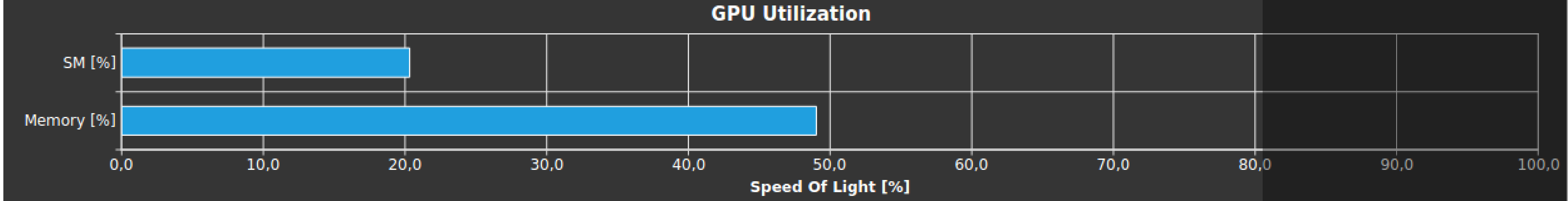
spline_and_spread: Old Version

Memory units more utilized than SM (Compute), but overall utilization is low
Nsight Compute hints that this is a latency issue, recommends further sections to check
We will still go through other sections for training purposes

GPU Speed Of Light ⚠ SOL Chart

SOL SM [%]	20,33	Duration [usecond]	66,24
SOL Memory [%]	49,04	Elapsed Cycles [cycle]	83.249
SOL TEX [%]	39,73	SM Active Cycles [cycle]	[illegible]
SOL L2 [%]	49,04	SM Frequency [cycle/nsecond]	[illegible]
SOL FB [%]	17,55	Memory Frequency [cycle/usecond]	[illegible]

Live

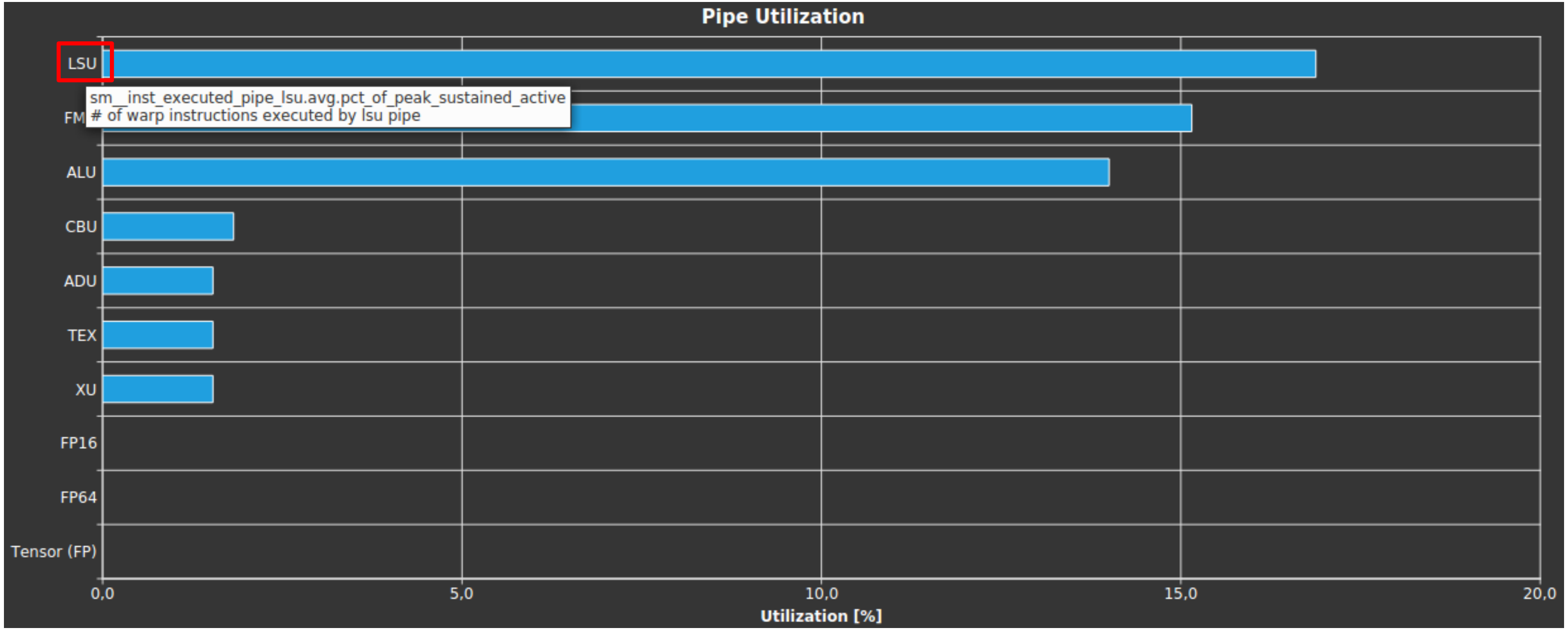


Recommendations

⚠ **Bottleneck** [Warning] This kernel exhibits low compute throughput and memory bandwidth utilization relative to the peak performance of this device. Achieved compute throughput and/or memory bandwidth below 60.0% of peak typically indicate latency issues. Look at `Scheduler Statistics` and `Warp State Statistics` for potential reasons.

spline_and_spread: Old Version

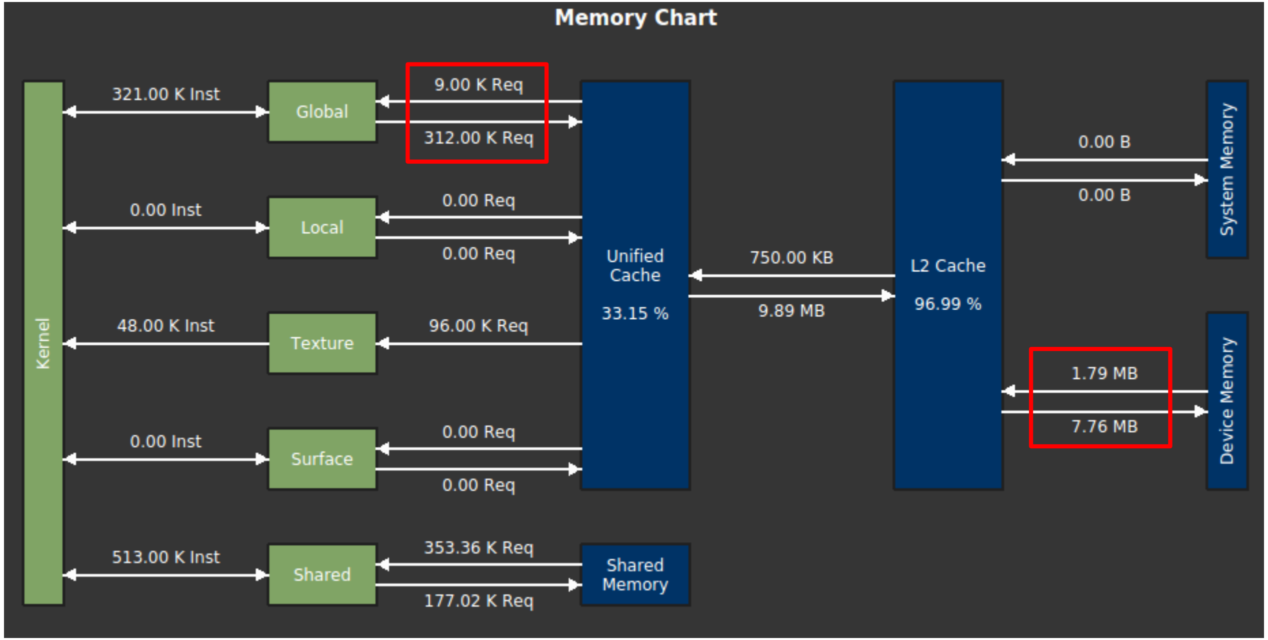
Highest utilized pipeline is LSU (Load Store Unit), indicating high load of memory load/store operations



spline_and_spread: Old Version

Memory chart shows that stores are much more common in this kernel, transferring ~10x as much data as reads

Since bandwidth is not saturated, it's likely frequent operations



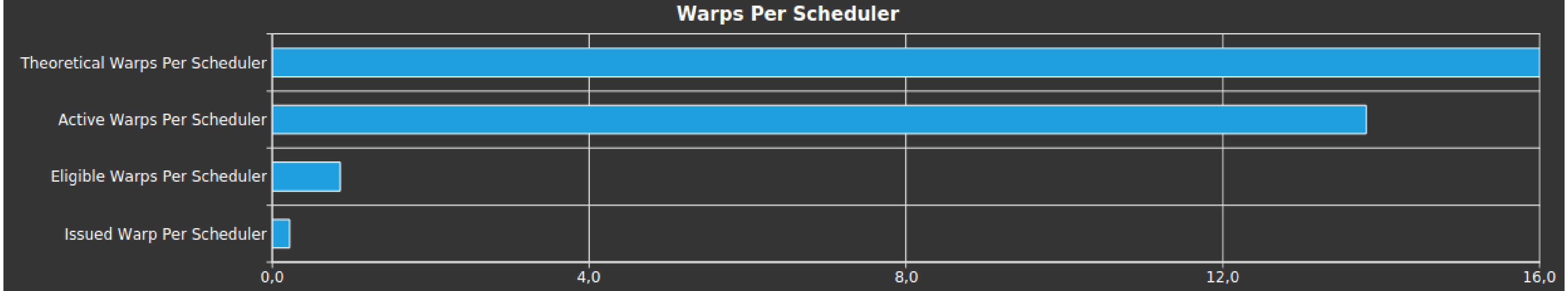
spline_and_spread: Old Version

We have many active warps available, but most of them are not eligible (and hence not issued) on average

The next section (Warp State Statistics) can indicate which stall reasons cause this

Scheduler Statistics

Active Warps Per Scheduler [warp]	13,81	Instructions Per Active Issue Slot [inst/cycle]	1
Eligible Warps Per Scheduler [warp]	0,86	No Eligible [%]	78,22
Issued Warp Per Scheduler	0,22	One or More Eligible [%]	21,78



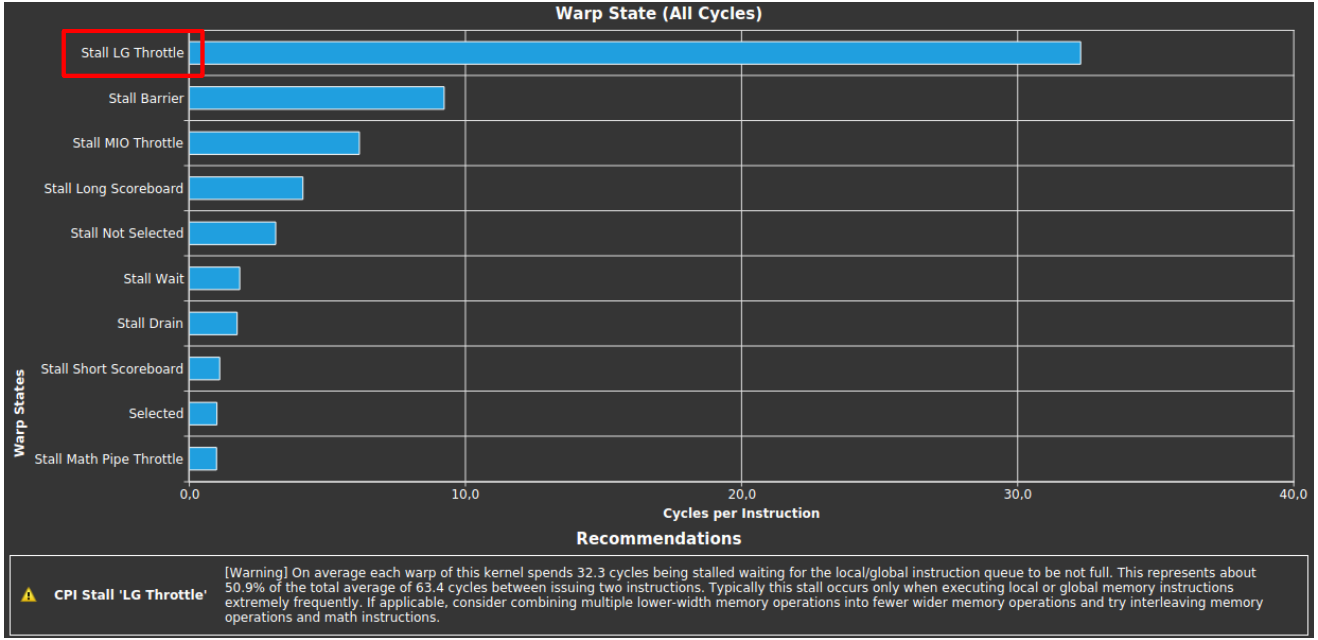
Recommendations

Issue Slot Utilization

[Warning] Every scheduler is capable of issuing one instruction per cycle, but for this kernel each scheduler only issues an instruction every 4.6 cycles. This might leave hardware resources underutilized and may lead to less optimal performance. Out of the maximum of 16 warps per scheduler, this kernel allocates an average of 13.81 active warps per scheduler, but only an average of 0.86 warps were eligible per cycle. Eligible warps are the subset of active warps that are ready to issue their next instruction. Every cycle with no eligible warp results in no instruction being issued and the issue slot remains unused. To increase the number of eligible warps either increase the number of active warps or reduce the time the active warps are stalled.

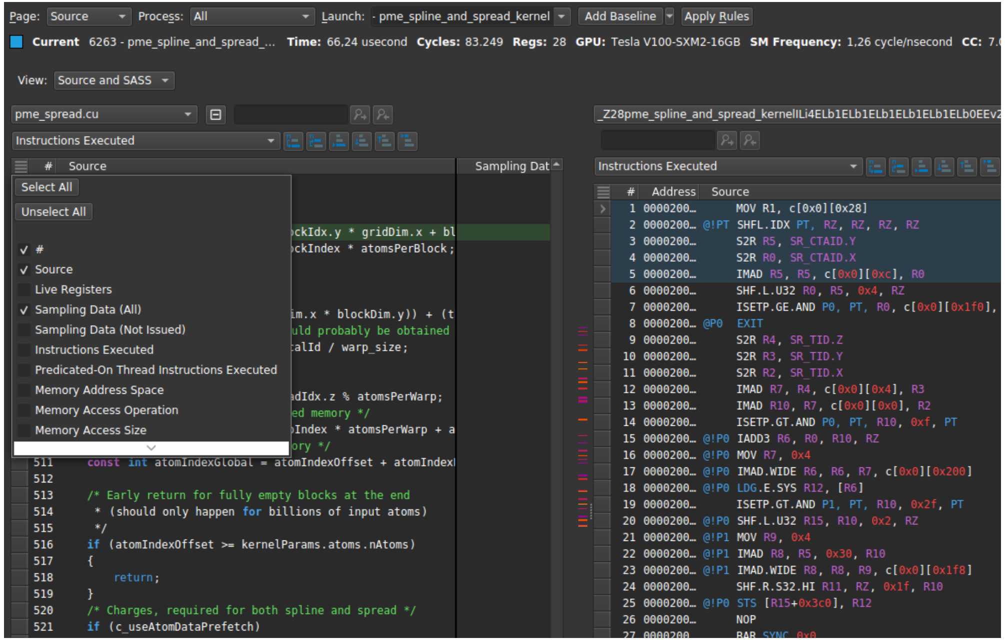
spline_and_spread: Old Version

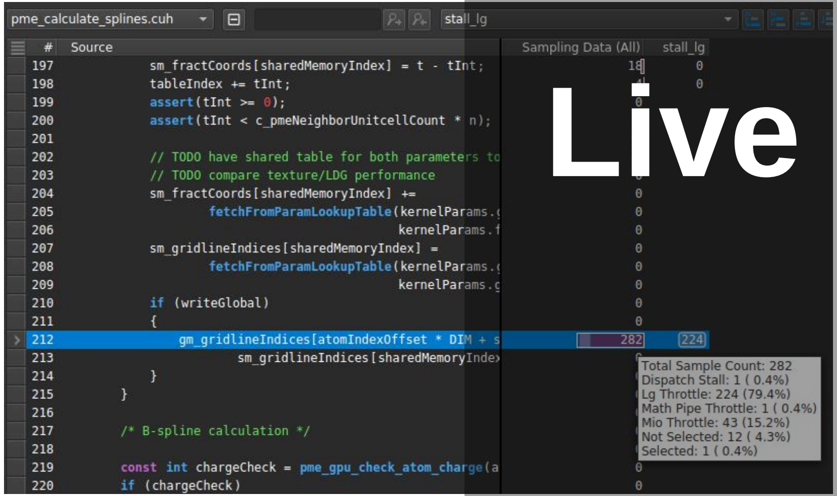
Most important stall reason (by far) is LG (local/global) Throttle
This indicates extremely frequent memory instructions, according to the guided analysis rule



spline_and_spread: Old Version

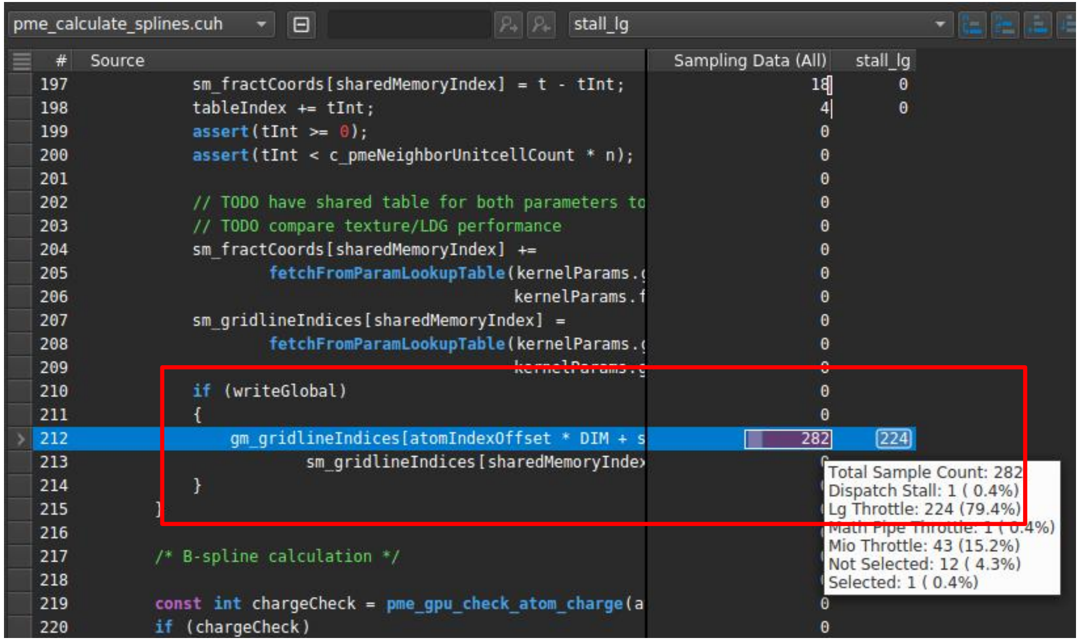
The samples locations of those stalls can be looked up on the Source page





spline_and_spread: Old Version

Disabling global memory writes to store temporary data (for the gather kernel) could reduce this latency issue
This implies that the gather kernel has to re-compute this data

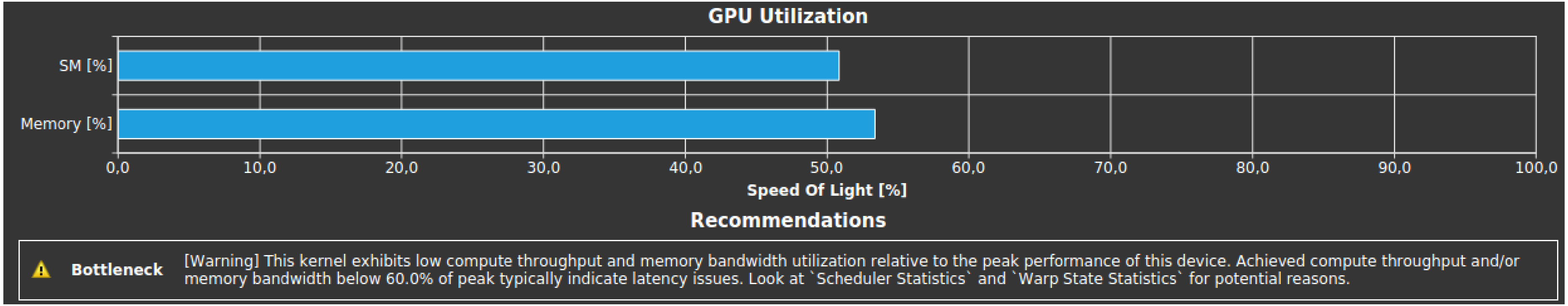


NVIDIA

gather: Old Version (overview)

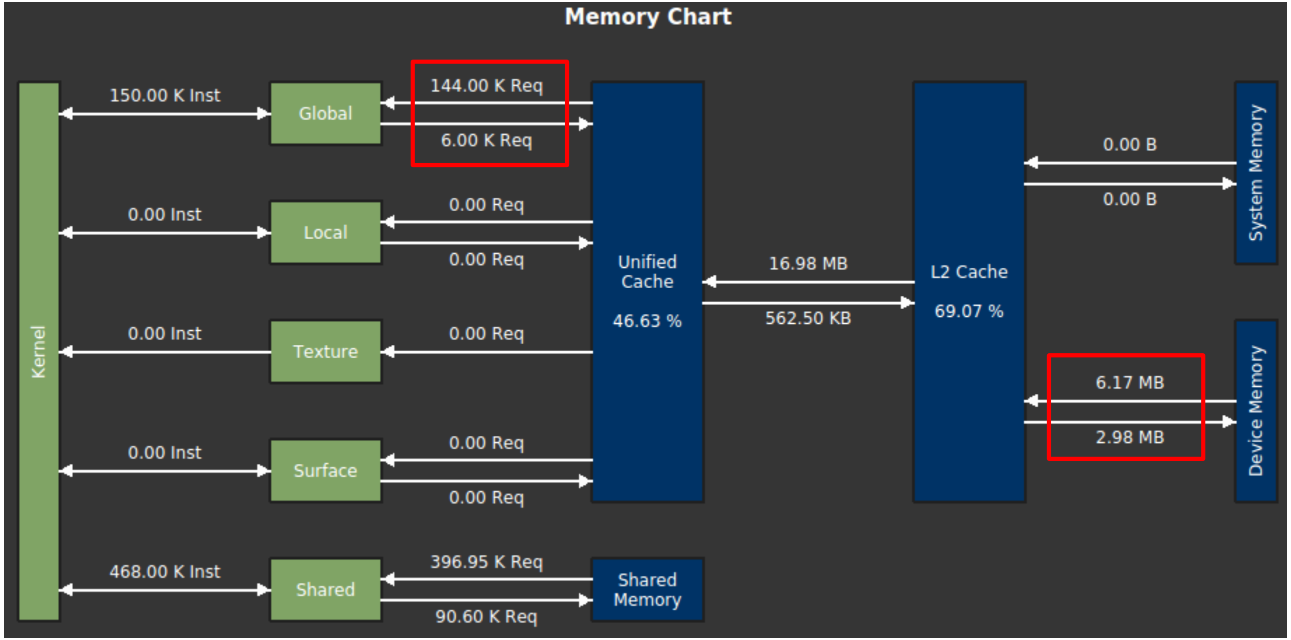
gather: Old Version

More balances compute/memory utilization, but also likely latency bound



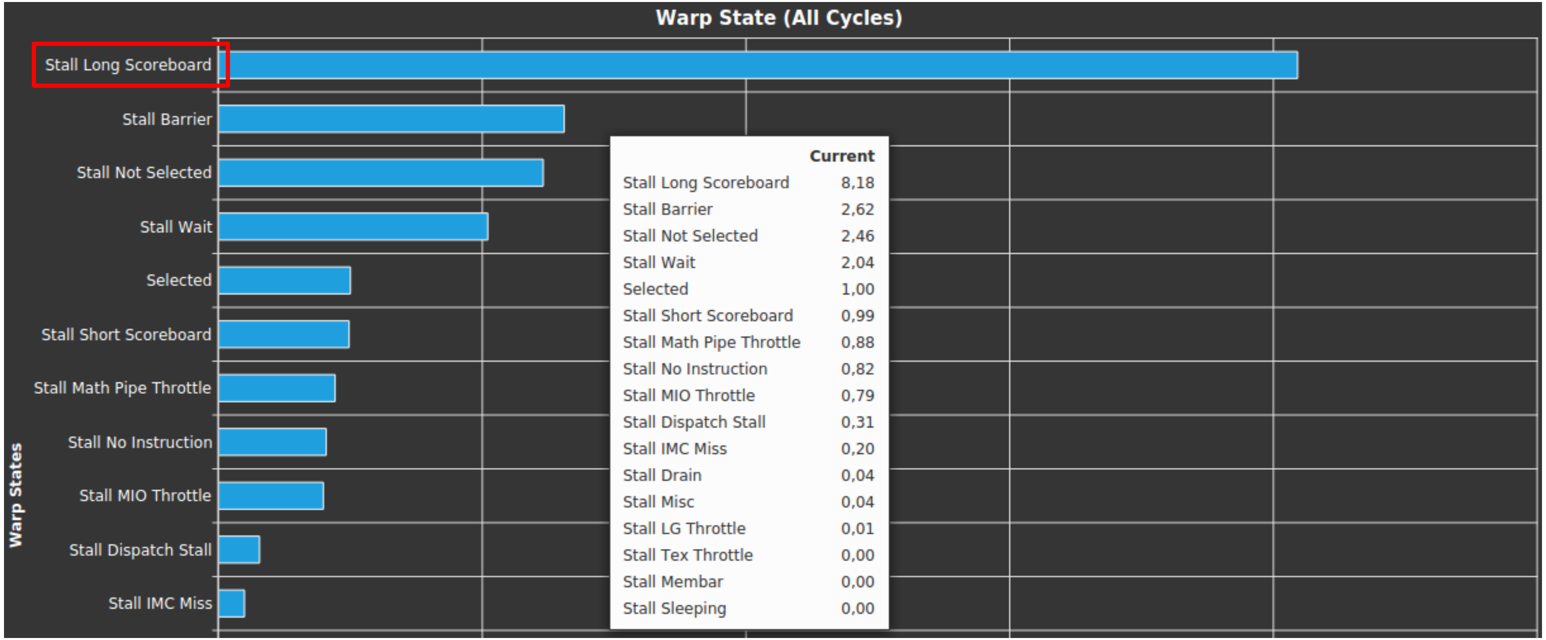
gather: Old Version

Reads temporary spline_and_spread kernel data from global memory
Therefore, much more load operations and data transfered in that direction



gather: Old Version

Long Scoreboard stalls cause most wasted cycles
These indicate waiting on local or global memory



GROMACS 2020
pme spread/gather
New Version

Code Changes

https://redmine.gromacs.org/projects/gromacs/repository/revisions/22118220401cee6f51d49c0a034e9fe5b4ba4260/diff?utf8=%E2%9C%93&type=sbs

Two new template arguments added to spread/gather kernels
Optimal kernel selected based on input data size
Disabled temp data storage in global memory for this analysis

pme_spline_and_spread_kernel	pme_gather_kernel
writeSplinesToGlobal control if we should write spline data to global memory *useOrderThreadsPerAtom** control if we should use order or order*order threads per atom	**readGlobal** control if we should read spline values from global memory *useOrderThreadsPerAtom** control if we should use order threads per atom (order*order used if false)

* not activated

spline_and_spread: New Version

spline_and_spread: New Version

Overall performance improvement is ~15% (fewer cycles)
Highest contributor appears to be the 54% reduced GPU DRAM throughput (SOL FB)

Current	62...	**Time:** 56,38 usecond	**Cycles:** 69.516	**Regs:** 20	**GPU:** Tesla V100-SXM2-16GB	**SM Frequency:** 1,23 cycle/nsecond	**CC:** 7.0	**Process:** [133280] gmx_mpi
unoptimized	62...	**Time:** 66,24 usecond	**Cycles:** 83.249	**Regs:** 28	**GPU:** Tesla V100-SXM2-16GB	**SM Frequency:** 1,26 cycle/nsecond	**CC:** 7.0	**Process:** [17712] gmx_mpi

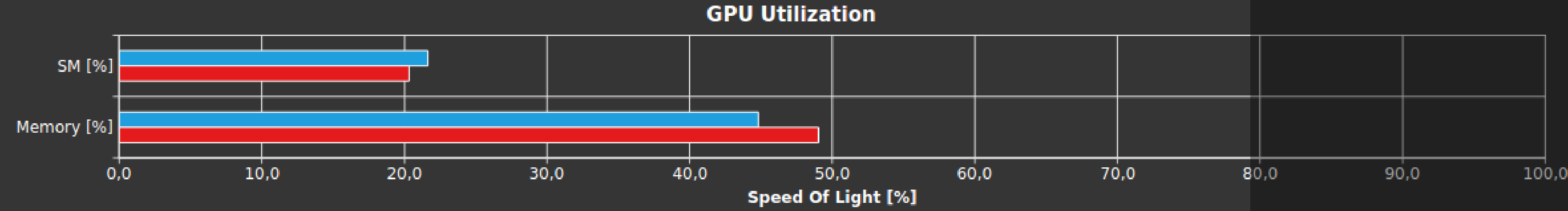
GPU Speed Of Light

All

Live

High-level overview of the utilization for compute and memory resources of the GPU. For each unit, the Speed Of Light (SOL) reports the achieved percentage of utiliz ion wit re pe t the theoretical maximum.

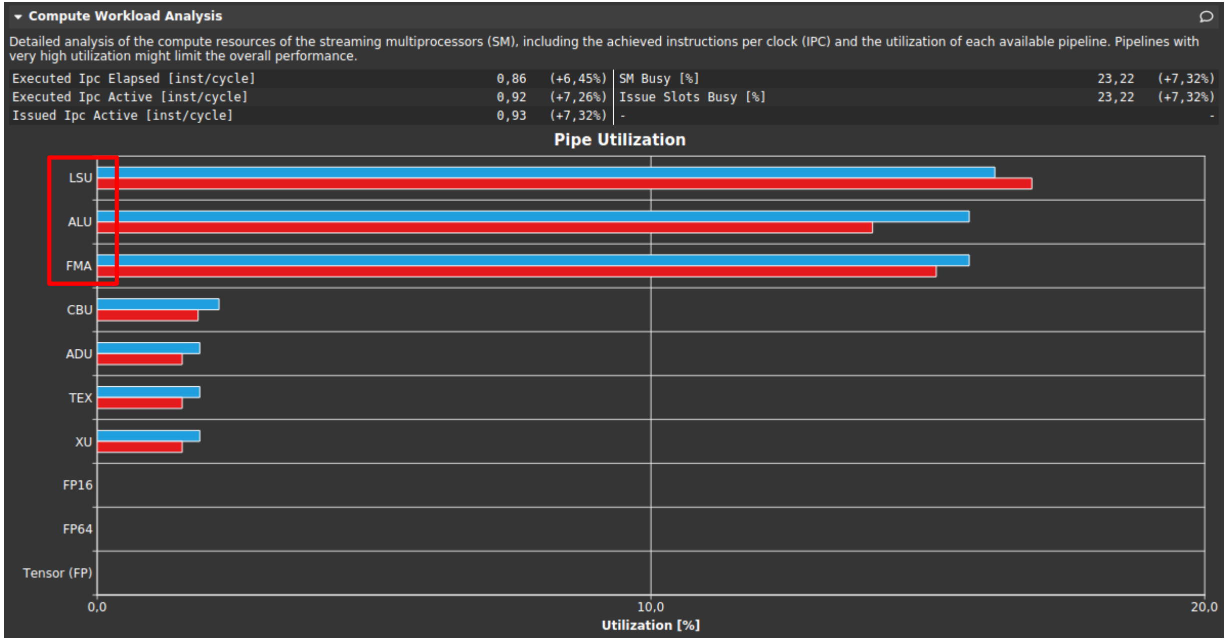
SOL SM [%]	21,65	(+6,51%)	Duration [usecond]	56,38	(-14,88%)
SOL Memory [%]	44,82	(-8,59%)	Elapsed Cycles [cycle]	69.516	(-16,50%)
SOL TEX [%]	37,02	(-6,82%)	SM Active Cycles [cycle]	64.771,97	(-17,18%)
SOL L2 [%]	44,82	(-8,59%)	SM Frequency [cycle/nsecond]	1,23	(-1,95%)
SOL FB [%]	7,99	(-54,46%)	Memory Frequency [cycle/usecond]	825,48	(-1,85%)



NVIDIA

spline_and_spread: New Version

Compute Workload Analysis shows slightly reduced usage of the load-store units pipeline in exchange for increased utilization of arithmetic pipelines (ALU, FMA)



spline_and_spread: New Version

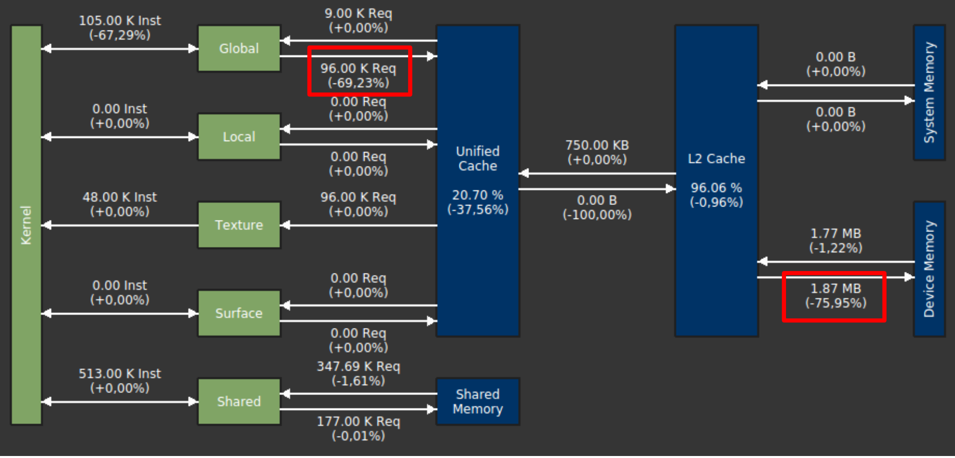
Reduced global store requests and data transfers to device memory

Memory Workload Analysis All

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

Memory Throughput [Gbyte/second]	67,56	(-55,30%)	Mem Busy [%]	31,15	(+9,92%)
L1 Hit Rate [%]	20,70	(-37,56%)	Max Bandwidth [%]	44,82	(-8,59%)
L2 Hit Rate [%]	96,06	(-0,96%)	Mem Pipes Busy [%]	15,12	(-4,69%)

Memory Chart

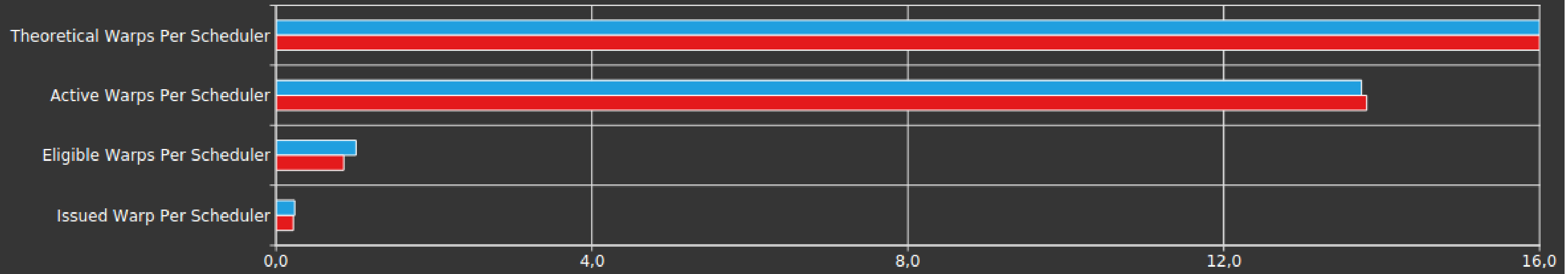


spline_and_spread: New Version

The eligible and issued warps/scheduler improved slightly (but are still quite low)

Active Warps Per Scheduler [warp]	13,74	(-0,49%)	Instructions Per Active Issue Slot [inst/cycle]	1	(+0,00%)
Eligible Warps Per Scheduler [warp]	1,01	(+18,15%)	No Eligible [%]	76,47	(-2,24%)
Issued Warp Per Scheduler	0,24	(+8,06%)	One or More Eligible [%]	23,53	(+8,06%)

Warps Per Scheduler

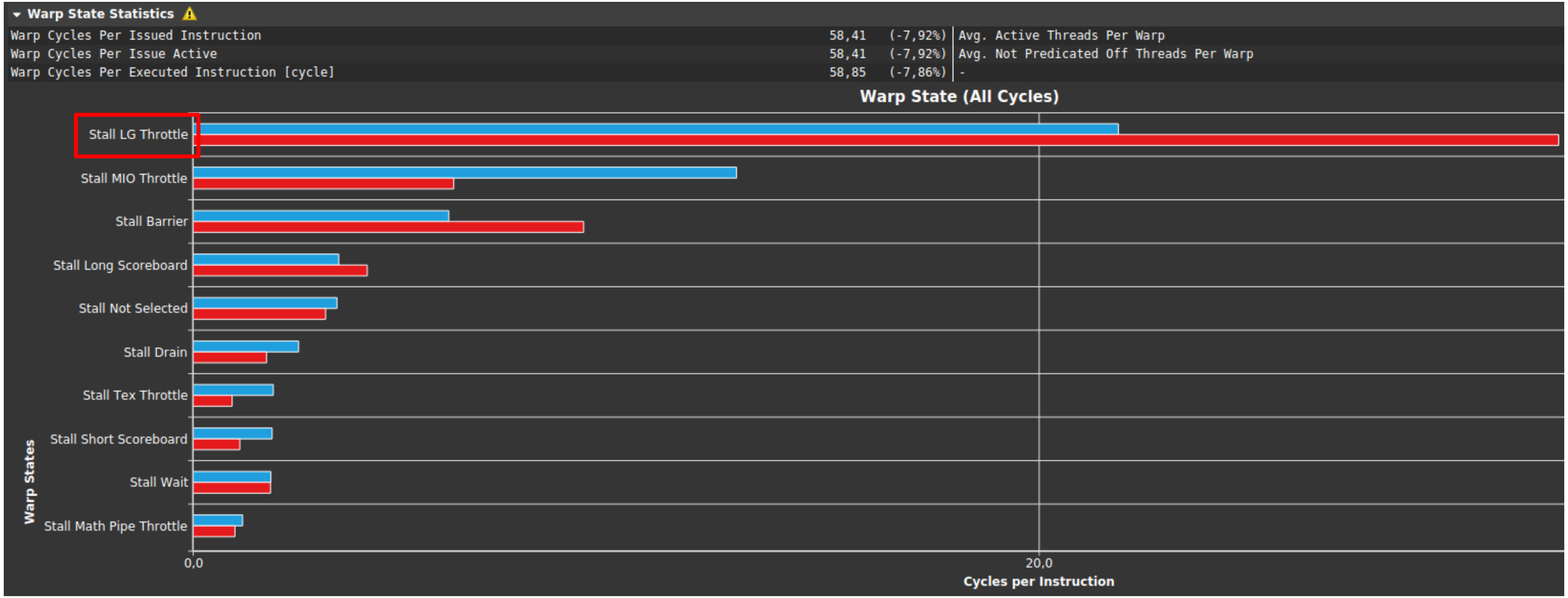


Recommendations

⚠ **Issue Slot Utilization**	[Warning] Every scheduler is capable of issuing one instruction per cycle, but for this kernel each scheduler only issues an instruction every 4.2 cycles. This might leave hardware resources underutilized and may lead to less optimal performance. Out of the maximum of 16 warps per scheduler, this kernel allocates an average of 13.74 active warps per scheduler, but only an average of 1.01 warps were eligible per cycle. Eligible warps are the subset of active warps that are ready to issue their next instruction. Every cycle with no eligible warp results in no instruction being issued and the issue slot remains unused. To increase the number of eligible warps either increase the number of active warps or reduce the time the active warps are stalled.

spline_and_spread: New Version

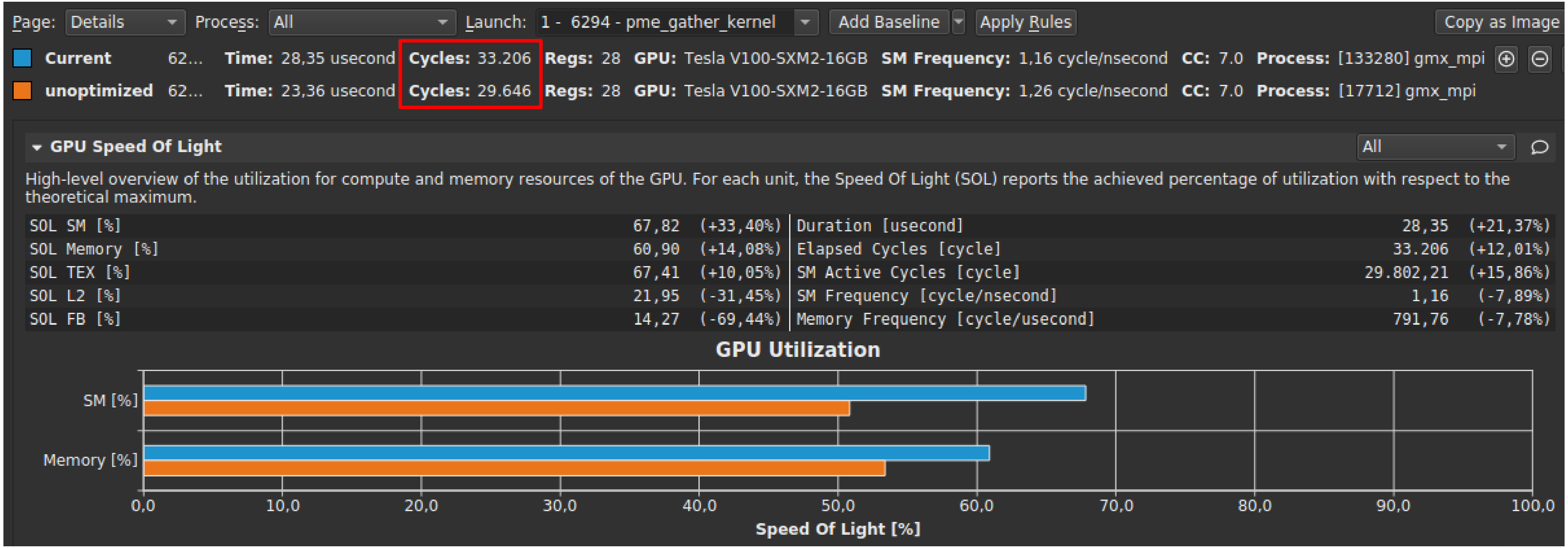
The improvement is due to reduced LG (local/global) Throttle stalls
(since we have fewer writes to memory)
Could be further reduced in a follow-up optimization



gather: New Version

gather: New Version

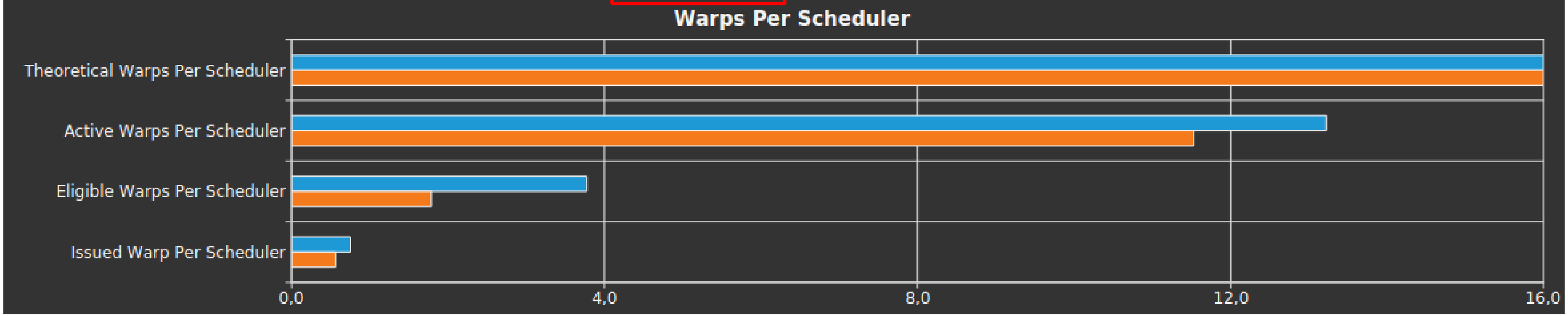
Performance decreased slightly compared with “unoptimized” version
The other individual sections allow us to identify what has changed in detail



gather: New Version

Recomputing instead of reading from global memory shows reduced cycles/inst for Long Scoreboard stalls...
...which translates to improved eligible and issued warps per scheduler

Active Warps Per Scheduler [warp]	13,23	(+14,72%)	Instructions Per Active Issue Slot [inst/cycle]	1	(+0,00%)
Eligible Warps Per Scheduler [warp]	3,77	(+111,61%)	No Eligible [%]	24,46	(-43,63%)
Issued Warp Per Scheduler	0,76	(+33,46%)	One or More Eligible [%]	75,54	(+33,46%)



gather: New Version

While the kernel executes instructions more efficiently now (higher IPC)...

Compute Workload Analysis

Detailed analysis of the compute resources of the streaming multiprocessors (SM), including the achieved instructions per clock (IPC) and the utilization of each available pipeline. Pipelines with very high utilization might limit the overall performance.

Executed Ipc Elapsed [inst/cycle]	2,69	(+33,26%)	SM Busy [%]	75,08	(+28,69%)
Executed Ipc Active [inst/cycle]	2,98	(+28,55%)	Issue Slots Busy [%]	75,08	(+28,69%)
Issued Ipc Active [inst/cycle]	3,00	(+28,69%)	-		-

...it also executes a lot more instructions in total (to re-compute values instead of loading them)

Instruction Statistics

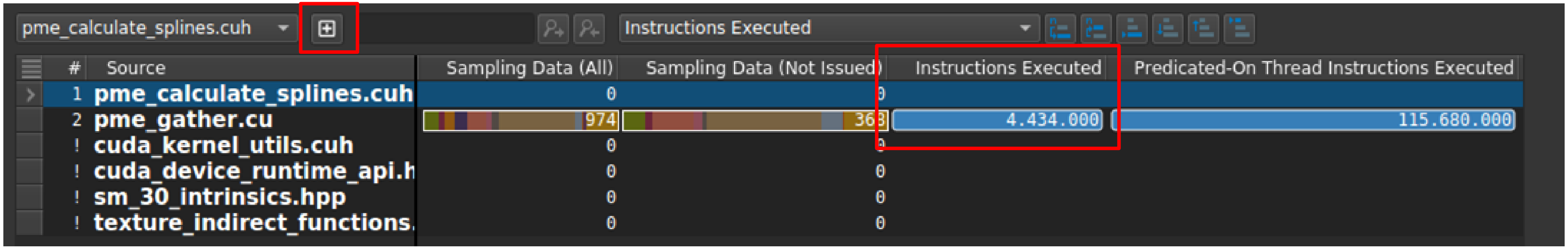
Statistics of the executed low-level assembly instructions (SASS). The instruction mix provides insight into the types and frequency of the executed instructions. A narrow mix of instruction types implies a dependency on few instruction pipelines, while others remain unused. Using multiple pipelines allows hiding latencies and enables parallel execution. Note that 'Instructions/Opcode' and 'Executed Instructions' are measured differently and can diverge if cycles are spent in system calls.

Executed Instructions [inst]	7.104.000	(+48,93%)	Avg. Executed Instructions Per Scheduler [inst]	22.200	(+48,93%)
Issued Instructions [inst]	7.160.121	(+49,09%)	Avg. Issued Instructions Per Scheduler [inst]	22.375,38	(+49,09%)

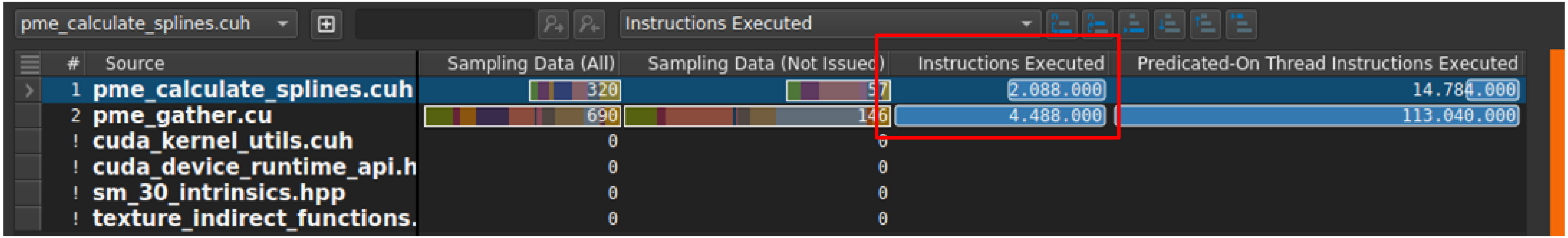
gather: Source Analysis

On the collapsed Source page, we can quickly identify where the new instructions originate

Old



New



New Version Summary

Overall, combined performance improved by ~10%
Use CSV export from CLI or UI to further analyze data in e.g. Excel

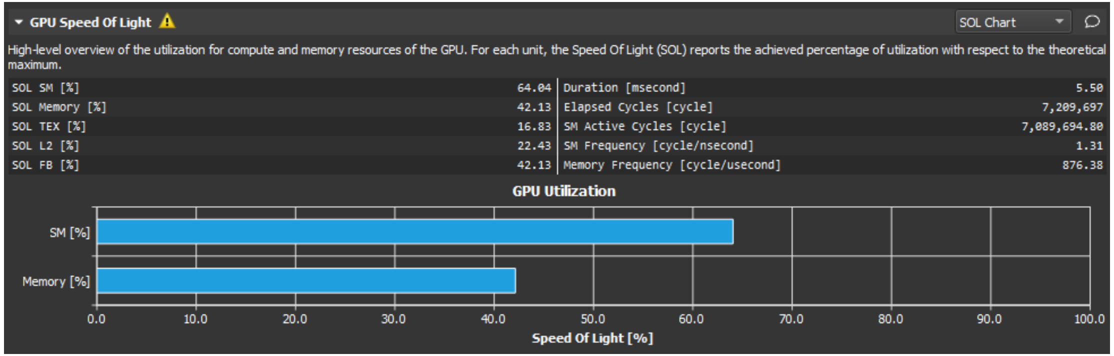
Function Name	Dema	Proces	Device I	Cycles [cycle]	SOL Memory [%]	SOL SM [%]
pme spline and spread kernel	**voi…**	**[17…**	**Tesla…**	**83.249**	**49,04**	**20,33**
pme gather kernel	voi…	[17…	Tesla…	29.646	53,38	50,84
pme spline and spread kernel	voi…	[17…	Tesla…	83.236	49,06	20,32
pme gather kernel	voi…	[17…	Tesla…	29.890	53,01	50,37
pme spline and spread kernel	voi…	[17…	Tesla…	83.624	48,80	20,23
pme gather kernel	voi…	[17…	Tesla…	29.051	54,58	51,86
pme spline and spread kernel	voi…	[17…	Tesla…	83.617	48,80	20,23
pme gather kernel	voi…	[17…	Tesla…	29.636	53,52	50,81
pme spline and spread kernel	voi…	[17…	Tesla…	83.070	49,12	20,37
pme gather kernel	voi…	[17…	Tesla…	29.591	53,68	50,87
pme spline and spread kernel	voi…	[17…	Tesla…	83.235	49,05	20,32
pme gather kernel	voi…	[17…	Tesla…	29.705	53,20	50,67
pme spline and spread kernel	voi…	[17…	Tesla…	82.910	49,19	20,40
pme gather kernel	voi…	[17…	Tesla…	29.233	54,32	51,50
pme spline and spread kernel	voi…	[17…	Tesla…	83.109	49,09	20,35
pme gather kernel	voi…	[17…	Tesla…	29.634	53,61	50,79
pme spline and spread kernel	voi…	[17…	Tesla…	83.299	48,98	20,31
pme gather kernel	voi…	[17…	Tesla…	30.130	52,62	49,97
pme spline and spread kernel	voi…	[17…	Tesla…	83.595	48,84	20,24
pme gather kernel	voi…	[17…	Tesla…	29.554	53,74	50,99

Function Name	Demang	Proces	Device	Cycles [cycle]	SOL Memory [%]	SOL SM [%]
pme spline and spread kernel	**void …**	**[13…**	**Tesl…**	**69.516**	**44,82**	**21,65**
pme gather kernel	void …	[13…	Tesl…	33.206	60,90	67,82
pme spline and spread kernel	void …	[13…	Tesl…	68.969	45,18	21,84
pme gather kernel	void …	[13…	Tesl…	33.497	60,29	67,15
pme spline and spread kernel	void …	[13…	Tesl…	68.516	45,44	21,97
pme gather kernel	void …	[13…	Tesl…	33.459	60,44	67,31
pme spline and spread kernel	void …	[13…	Tesl…	69.197	45,00	21,75
pme gather kernel	void …	[13…	Tesl…	34.020	59,38	66,15
pme spline and spread kernel	void …	[13…	Tesl…	70.337	44,29	21,39
pme gather kernel	void …	[13…	Tesl…	33.389	60,50	67,42
pme spline and spread kernel	void …	[13…	Tesl…	69.615	44,73	21,62
pme gather kernel	void …	[13…	Tesl…	33.558	60,23	67,06
pme spline and spread kernel	void …	[13…	Tesl…	69.261	44,96	21,73
pme gather kernel	void …	[13…	Tesl…	33.457	60,46	67,32
pme spline and spread kernel	void …	[13…	Tesl…	68.927	45,16	21,84
pme gather kernel	void …	[13…	Tesl…	33.874	59,62	66,40
pme spline and spread kernel	void …	[13…	Tesl…	69.266	44,95	21,73
pme gather kernel	void …	[13…	Tesl…	33.221	60,91	67,82
pme spline and spread kernel	void …	[13…	Tesl…	68.452	45,50	22,00
pme gather kernel	void …	[13…	Tesl…	33.432	60,54	67,32

Customize Data Collection and Analysis

Customize Sections

```
Identifier: "SpeedOfLight"
DisplayName: "GPU Speed Of Light"
Description: "High-level overview of ..."
Order: 10
Sets {
  Identifier: "default"
}
Sets {
  Identifier: "full"
}
Header {
  Metrics {
    Label: "SOL SM"
    Name: "sm__throughput.avg.
pct_of_peak_sustained_elapsed"
  }
  Metrics {
    Label: "Duration"
    Name: "gpu__time_duration.sum"
  }
  Metrics {
    Label: "SOL Memory"
    Name: "gpu__compute_memory_throughput.avg.
pct_of_peak_sustained_elapsed"
  }
  Metrics {
    Label: "Elapsed Cycles"
    Name: "gpc__cycles_elapsed.max"
  }
...
```

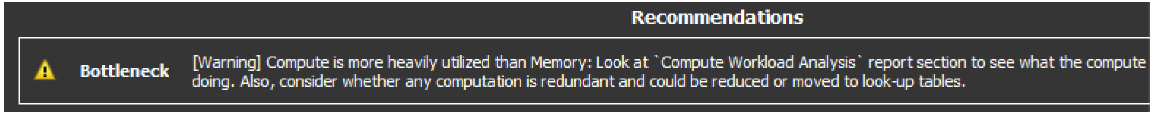


Metrics collection
Metric presentation
- Tables
- Charts
- Source page correlation
- Details page ordering

Section set association

Customize Rules

```
import NvRules
import math

def get_identifier():
    return "SOLBottleneck"

def get_section_identifier():
    return "SpeedOfLight"

def apply(handle):
    ctx = NvRules.get_context(handle)
    action = ctx.range_by_idx(0).action_by_idx(0)
    fe = ctx.frontend()

    num_waves =
action.metric_by_name("launch__waves_per_multiprocessor")
.as_double()
    smSolPct = action.metric_by_name("sm__throughput.avg
.pct_of_peak_sustained_elapsed").as_double()
    memSolPct =
action.metric_by_name("gpu__compute_memory_throughput.avg
.pct_of_peak_sustained_elapsed").as_double()

    balanced_threshold = 10
    latency_bound_threshold = 60
    no_bound_threshold = 80
    waves_threshold = 1
...
```



Python rules accessing collected data

Available in UI and CLI

Performance analysis guidance

- By NVIDIA
- By your own experts

Allow users better understanding of recommendations

In next version: dynamic report navigation

Conclusion

Known Issues/Outlook

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

Outlook for next version

Improved multi-process/MPI support

Parity with nvprof report name placeholders (process ID, env var, running number)

Better kernel name demangler

Improved memory workload analysis tables

Dynamic report navigation

Uncoalesced memory rules

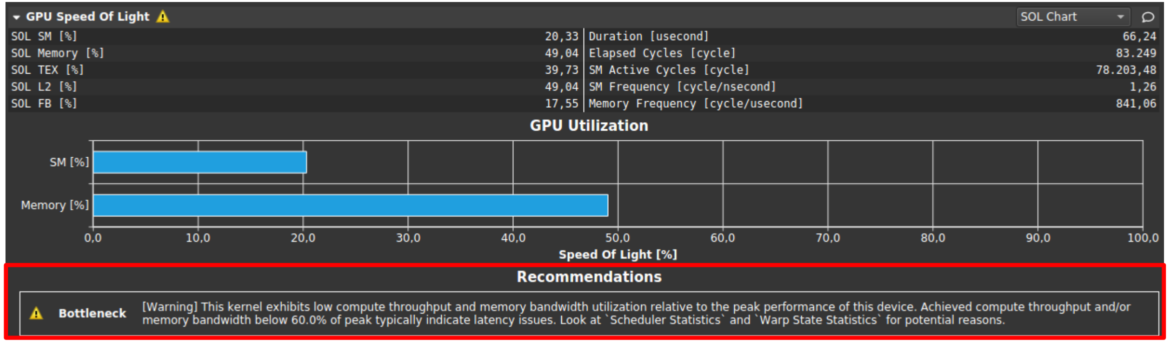
Conclusion

Nsight Compute enables detailed kernel analysis

Rules give guidance on optimization opportunities and help metric understanding

Limit sections/metrics to what is required when overhead is a concern

Still requires level of hardware understanding to fully utilize the tool - pay attention to rule results



THANK YOU!

Download	https://developer.nvidia.com/nsight-compute (can we newer than toolkit version)
Documentation	https://docs.nvidia.com/nsight-compute (and local with the tool)
Forums	https://devtalk.nvidia.com
Further Training	**Blue Waters Seminar** https://bluewaters.ncsa.illinois.edu/webinars/petascale-computing/nsight-compute **GTC 2019** https://developer.nvidia.com/gtc/2019/video/S9345 **Blog posts** https://devblogs.nvidia.com/using-nsight-compute-to-inspect-your-kernels/ https://devblogs.nvidia.com/migrating-nvidia-nsight-tools-nvvp-nvprof/


NVIDIA®