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
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
Nsight Compute

Source/PTX/SASS analysis and correlation

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

Metric heatmap
Nsight Compute

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
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/nsight-compute/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" …
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
Consider profiling only a single rank, e.g. using a wrapper script

```bash
#!/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
```
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


<table>
<thead>
<tr>
<th>Differences</th>
<th>Missing Features (in progress)</th>
</tr>
</thead>
<tbody>
<tr>
<td>New metric names and many more metrics <a href="https://docs.nvidia.com/nsight-compute/NsightComputeCli/index.html#nvprof-metric-comparison">https://docs.nvidia.com/nsight-compute/NsightComputeCli/index.html#nvprof-metric-comparison</a></td>
<td>Application replay</td>
</tr>
<tr>
<td>Cache flush and clock control enabled by default for deterministic data collection</td>
<td>No NVLink metrics</td>
</tr>
<tr>
<td>Customizable</td>
<td>No trace - use <a href="https://devblogs.nvidia.com/migrating-nvidia-nsight-tools-nvvp-nvprof/">Nsight Systems</a></td>
</tr>
<tr>
<td></td>
<td>No MPS support</td>
</tr>
</tbody>
</table>
GROMACS 2020
pme spread/gather
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.
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.
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.
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.
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.
gather: Old Version (overview)
gather: Old Version

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

[Graph showing GPU Utilization with SM [%] and Memory [%] bars.]

[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.]
gather: Old Version

Reads temporary spline_and_spread kernel data from global memory
Therefore, much more load operations and data transferred in that direction
Long Scoreboard stalls cause most wasted cycles
These indicate waiting on local or global memory
GROMACS 2020
pme spread/gather
New Version
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

<table>
<thead>
<tr>
<th>pme_spline_and_spread_kernel</th>
<th>pme_gather_kernel</th>
</tr>
</thead>
<tbody>
<tr>
<td>writeSplinesToGlobal</td>
<td>readGlobal</td>
</tr>
<tr>
<td>control if we should write spline data to global memory</td>
<td>control if we should read spline values from global memory</td>
</tr>
<tr>
<td>useOrderThreadsPerAtom*</td>
<td>useOrderThreadsPerAtom*</td>
</tr>
<tr>
<td>control if we should use order or order*order threads per atom</td>
<td>control if we should use order threads per atom (order*order used if false)</td>
</tr>
</tbody>
</table>

* 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)
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
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 (+0.08%) |
| Eligible Warps Per Scheduler [warp] | 1.81 (+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**

**Theoretical Warps Per Scheduler**

**Active Warps Per Scheduler**

**Eligible Warps Per Scheduler**

**Issued Warp Per Scheduler**

**Recommendations**

*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 1.81 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.
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
Performance decreased slightly compared with “unoptimized” version. The other individual sections allow us to identify what has changed in detail.
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
While the kernel executes instructions more efficiently now (higher IPC)...

...it also executes a lot more instructions in total (to re-compute values instead of loading them)
On the collapsed Source page, we can quickly identify where the new instructions originate.

**Old**

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

    ...
Conclusion
Known Issues/Outlook

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