

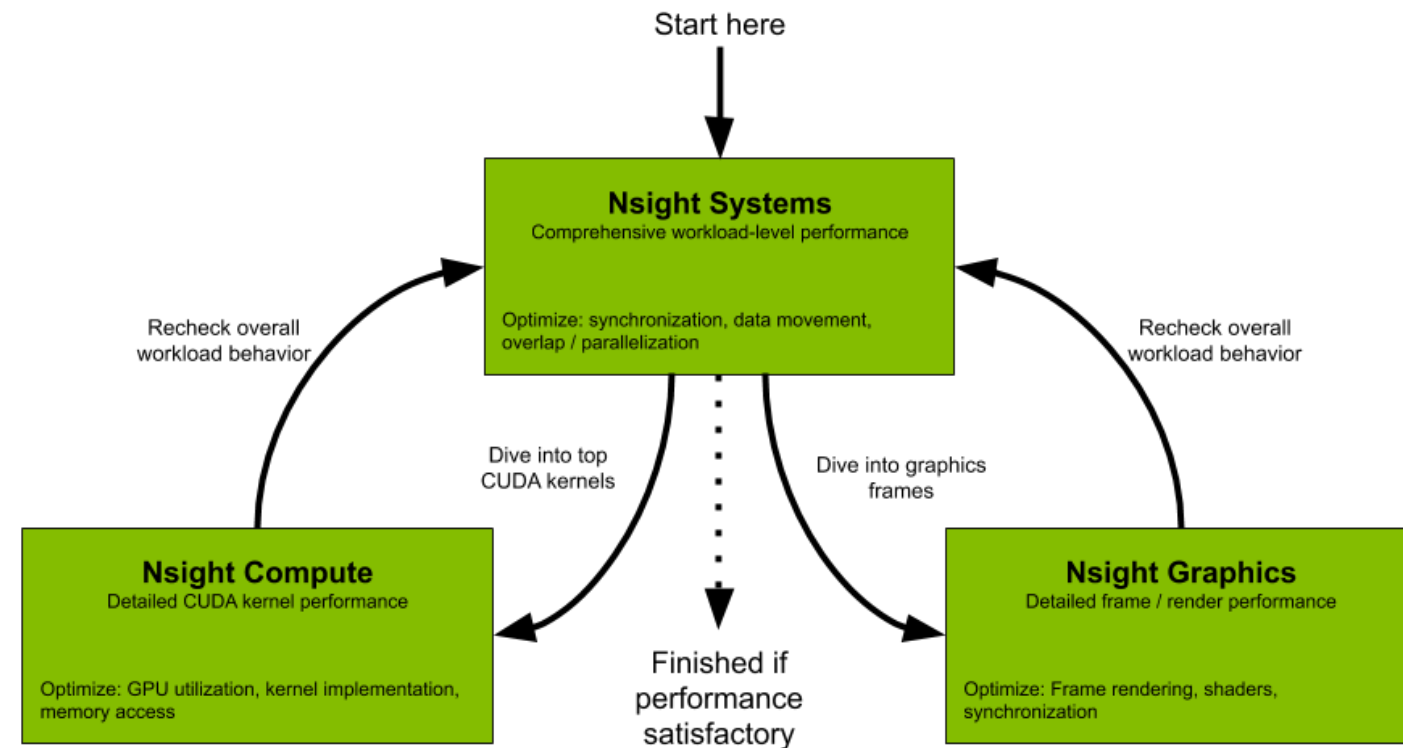
ALCF DevSession, February 23rd, 2022

Performance Analysis using Nsight Systems/Compute Tools

JaeHyuk Kwack (jkwack@anl.gov)
ALCF Performance Engineering Group

NVIDIA Tools overview

- Nsight Systems
 - A system-wide visualization of an application performance
 - To optimize bottlenecks to scale efficiently across CPUs and GPUs on ThetaGPU
- Nsight Compute
 - An interactive kernel profiler for applications
 - Providing detailed performance metrics and API debugging via a user interface and command line tool
 - Providing customizable and data-driven user interface and metric collection that can be extended with analysis scripts for post-processing results
- Nsight Graphics
 - A stand-alone tool for graphics applications



Credit: NVIDIA (<https://developer.nvidia.com/tools-overview>)

Step-by-step guide on ThetaGPU (1/2)

- Common part on ThetaGPU

- Build your application for ThetaGPU

- Submit your job script to ThetaGPU or start an interactive job mode on ThetaGPU as follows:

```
$ module load cobalt/cobalt-gpu
```

```
$ qsub -I -n 1 -t 30 -q full-node -A {your_project}
```

- Nsight Systems

- Run your application with Nsight Systems as follows:

```
$ nsys profile -o {output_filename} --stats=true ./{your_application}
```

- Nsight Compute

- Run your application with Nsight Compute

```
$ ncu --set detailed -k {kernel_name} -o {output_filename} ./{your_application}
```

Or,

```
$ ncu --set detailed -k regex:"kernel1|kernel2" -o {output_filename}  
./{your_application}
```

- Remark: W/o -o option, Nsight Compute provides performance data as a standard output

Step-by-step guide on ThetaGPU (2/2)

- Post-processing the profiled data

- Post-processing via CLI

- `$ nsys stats {output_filename}.qdrep`

- `$ ncu -i {output_filename}.ncu-rep`

- Post-processing on your local system via GUI

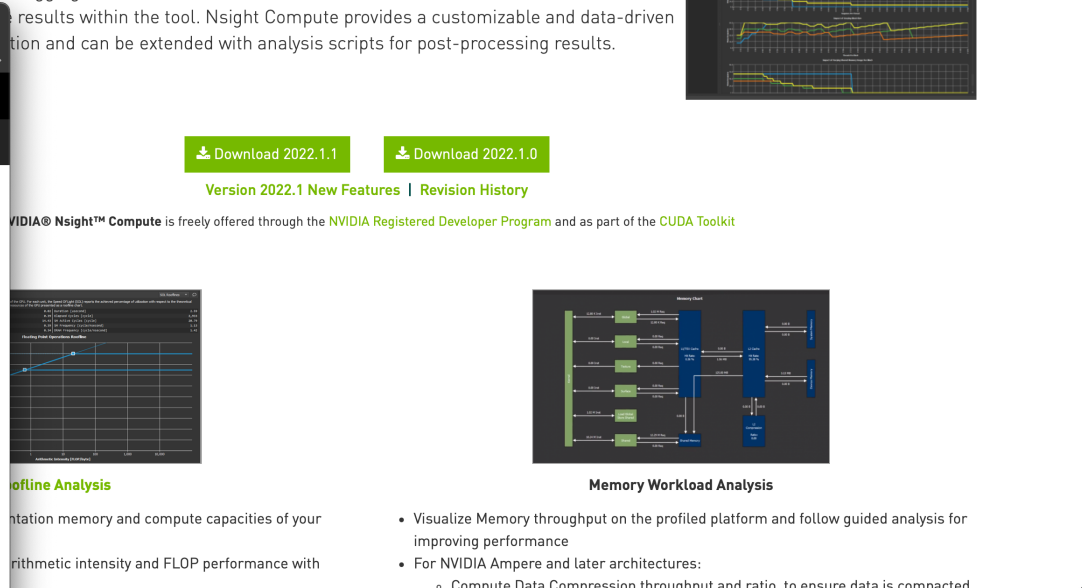
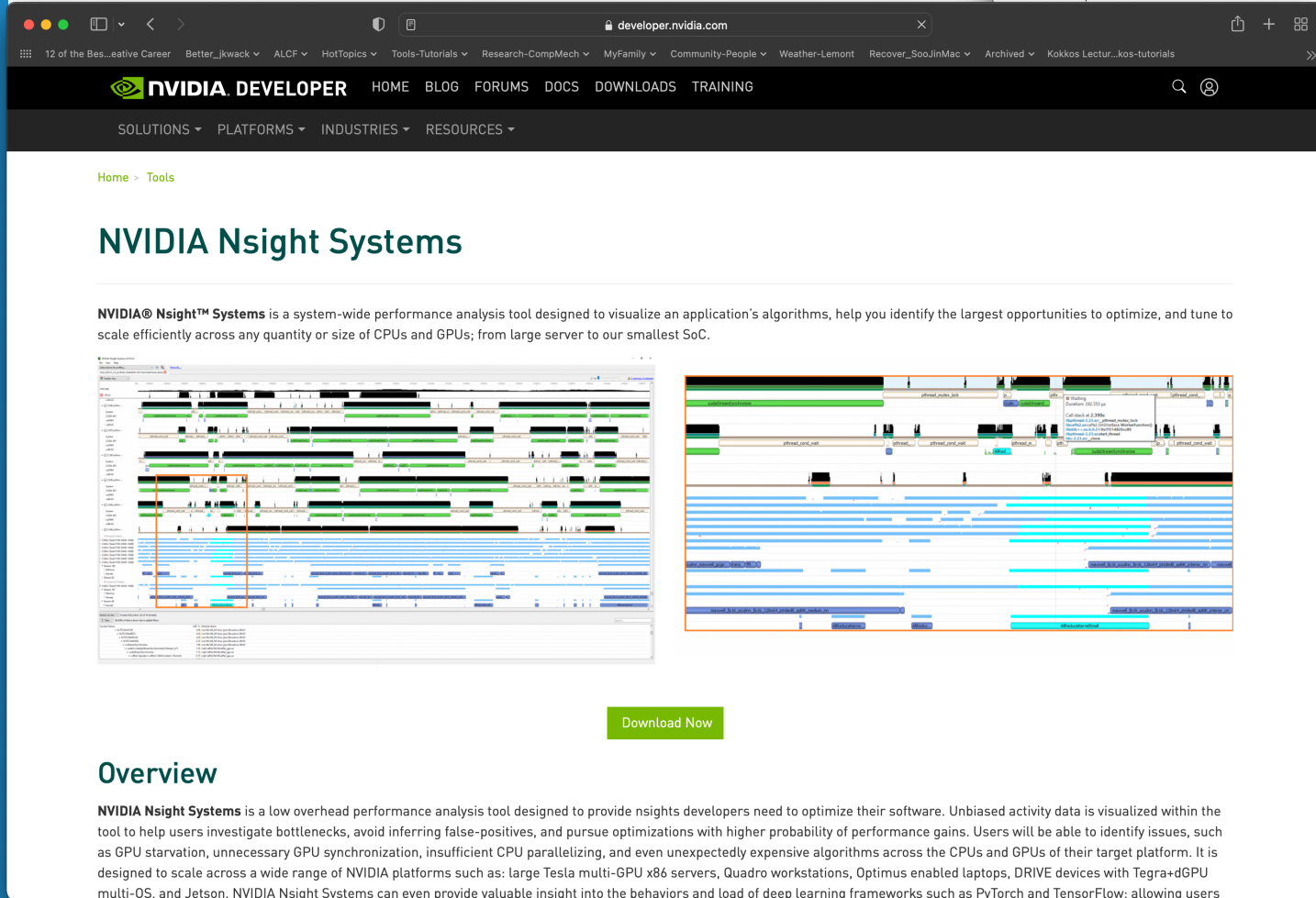
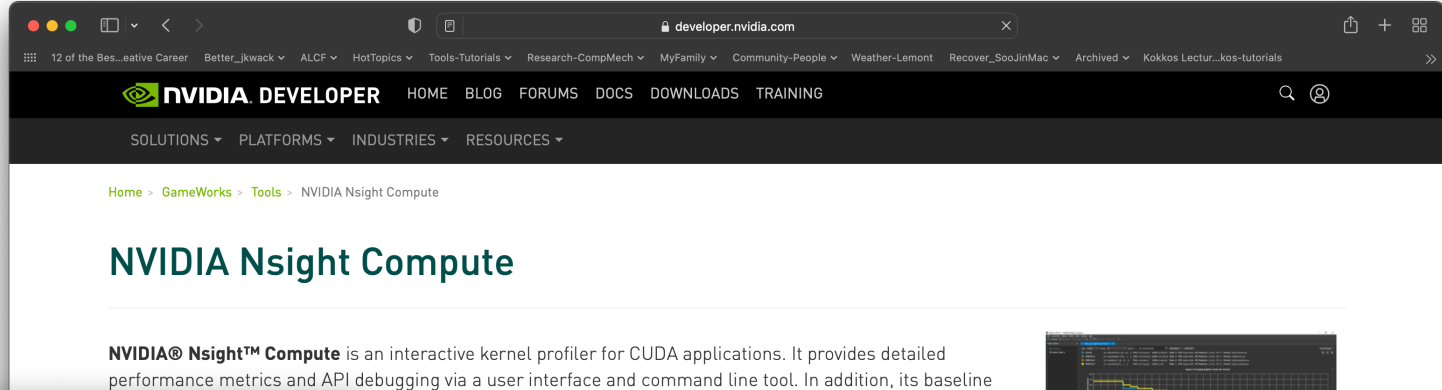
- Install NVIDIA Nsight Systems and NVIDIA Nsight Compute after downloading both of them from the [NVIDIA Developer Zone](#).
 - Download nsys output files (i.e., ending with .qdrep and .sqlite) to your local system, and then open them with NVIDIA Nsight Systems on your local system.
 - Download ncu output files (i.e., ending with .ncu-rep) to your local system, and then open them with NVIDIA Nsight Compute on your local system.

- More options for performance analysis with Nsight Systems and Nsight Compute

- `$ nsys --help`

- `$ ncu --help`

NVIDIA Developer Zone



<https://developer.nvidia.com/nsight-systems>
<https://developer.nvidia.com/nsight-compute>

Case 1: Stream benchmark



Stream Benchmark

- BabelStream: stream benchmark implemented in multiple programming models
 - <https://github.com/UoB-HPC/BabelStream>
 - **CUDA**, OpenMP, SYCL, HIP, OpenCL, Kokkos, RAJA, and so on
- Five kernels to measure memory bandwidth for a given array (size: N)

kernal	source	FLOP	Bytes	Arith. Intensity	Instruction
copy	<code>c[i] = a[i];</code>	0	16N	0	
mul	<code>b[i] = scalar * c[i];</code>	N	16N	0.063	Mul
add	<code>c[i] = a[i] + b[i];</code>	N	24N	0.042	ADD
triad	<code>a[i] = b[i] + scalar * c[i]</code>	2N	24N	0.083	FMA
dot	<code>tb_sum[local_i] += a[i] * b[i];</code>	2N	32N	0.063	FMA

```
template <typename T>
__global__ void add_kernel(const T * a, const T * b, T * c)
{
    const int i = blockDim.x * blockIdx.x + threadIdx.x;
    c[i] = a[i] + b[i];
}
```

```
template <typename T>
__global__ void triad_kernel(T * a, const T * b, const T * c)
{
    const T scalar = startScalar;
    const int i = blockDim.x * blockIdx.x + threadIdx.x;
    a[i] = b[i] + scalar * c[i];
}
```

Nsight Systems with Stream benchmark

```
$ nsys profile -o JKreport-nsys-BableStream --stats=true ./cuda-stream
```

```
...
```

```
Generating CUDA API Statistics...
```

```
CUDA API Statistics (nanoseconds)
```

Time(%)	Total Time	Calls	Average	Minimum	Maximum	Name
44.8	280504347	4	70126086.8	1050249	276881346	cudaMalloc
31.4	196878210	401	490968.1	381542	600948	cudaDeviceSynchronize
22.4	140280462	103	1361946.2	436597	32339232	cudaMemcpy
1.0	6263864	4	1565966.0	1236542	1884610	cudaFree
0.4	2729558	501	5448.2	4970	36269	cudaLaunchKernel

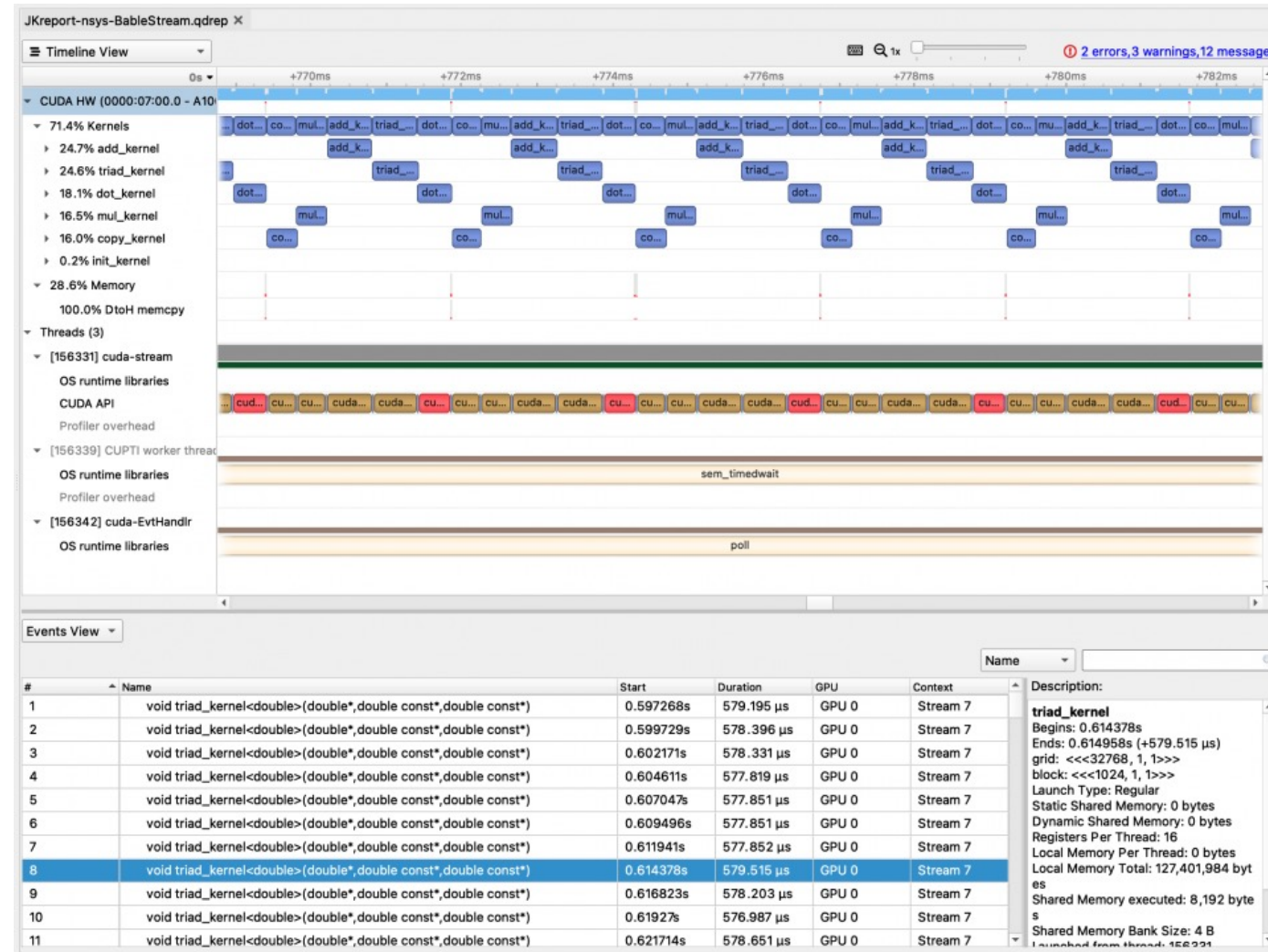
```
Generating CUDA Kernel Statistics...
```

```
CUDA Kernel Statistics (nanoseconds)
```

Time(%)	Total Time	Instances	Average	Minimum	Maximum	Name
24.7	58518170	100	585181.7	580347	594395	void add_kernel<double>(double const*, double const*, double*)
24.6	58312184	100	583121.8	576987	595067	void triad_kernel<double>(double*, double const*, double const*)
18.1	42942748	100	429427.5	419548	438333	void dot_kernel<double>(double const*, double const*, double*, int)
16.5	39062588	100	390625.9	388733	392125	void mul_kernel<double>(double*, double const*)
16.0	37980930	100	379809.3	376541	392925	void copy_kernel<double>(double const*, double*)
0.2	521628	1	521628.0	521628	521628	void init_kernel<double>(double*, double*, double*, double, double, double)

```
...
```


Nsight Systems with Stream benchmark



Nsight Compute with Stream benchmark

```
$ ncu --set detailed -k reg:"triad|add" -o JKreport-ncu_detailed-triad-add-BableStream ./cuda-stream
BabelStream
Version: 3.4
Implementation: CUDA
Running kernels 100 times
Precision: double
Array size: 268.4 MB (=0.3 GB)
Total size: 805.3 MB (=0.8 GB)
==PROF== Connected to process 166971 (/gpfs/mira-home/jkwack/HPC_benchmarks/BabelStream/JK_thetaGPU/cuda-stream)
Using CUDA device A100-SXM4-40GB
Driver: 11000
==PROF== Profiling "add_kernel": 0%...50%...100% - 18 passes
==PROF== Profiling "triad_kernel": 0%...50%...100% - 18 passes
==PROF== Profiling "add_kernel": 0%...50%...100% - 18 passes
==PROF== Profiling "triad_kernel": 0%...50%...100% - 18 passes
...
==PROF== Profiling "triad_kernel": 0%...50%...100% - 18 passes
Function  MBytes/sec  Min (sec)  Max      Average
Copy      1328115.853  0.00040   0.00042  0.00041
Mul       1302136.580  0.00041   0.00043  0.00042
Add       945.574     0.85166   1.15119  0.91029
Triad     943.893     0.85318   1.12513  0.95746
Dot       842990.134  0.00064   0.00074  0.00067
==PROF== Disconnected from process 2627262
...
```


Nsight Compute w/ Stream

Page: Details Launch: 0 - 867 - triad_kernel Add Baseline Apply Rules Copy as Image

Current 867 ... Time: 572.19 usecond Cycles: 622,968 Regs: 16 GPU: A100-SXM4-40GB SM Frequency: 1.09 cycle/nsecond CC: 8.0 Process: [166971] cuda-stream

GPU Speed Of Light

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 utilization with respect to the theoretical maximum. High-level overview of the utilization for compute and memory resources of the GPU presented as a roofline chart.

SOL SM [%]	6.23	Duration [usecond]	572.19
SOL Memory [%]	89.75	Elapsed Cycles [cycle]	622968
SOL L1/TEX Cache [%]	15.70	SM Active Cycles [cycle]	618515.41
SOL L2 Cache [%]	69.01	SM Frequency [cycle/nsecond]	1.09
SOL DRAM [%]	89.75	DRAM Frequency [cycle/nsecond]	1.21

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]	0.22	SM Busy [%]	5.50
Executed Ipc Active [inst/cycle]	0.22	Issue Slots Busy [%]	5.50
Issued Ipc Active [inst/cycle]	0.22		

Memory Workload Analysis

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).

Memory Throughput [Tbyte/second]	1.39	Mem Busy [%]	51.96
L1/TEX Hit Rate [%]	0	Max Bandwidth [%]	89.75
L2 Hit Rate [%]	50.06	Mem Pipes Busy [%]	6.23
L2 Compression Success Rate [%]	0	L2 Compression Ratio	0

Scheduler Statistics

Summary of the activity of the schedulers issuing instructions. Each scheduler maintains a pool of warps that it can issue instructions for. The upper bound of warps in the pool (Theoretical Warps) is limited by the launch configuration. On every cycle each scheduler checks the state of the allocated warps in the pool (Active Warps). Active warps that are not stalled (Eligible Warps) are ready to issue their next instruction. From the set of eligible warps the scheduler selects a single warp from which to issue one or more instructions (Issued Warp). On cycles with no eligible warps, the issue slot is skipped and no instruction is issued. Having many skipped issue slots indicates poor latency hiding.

Active Warps Per Scheduler [warp]	13.95	Instructions Per Active Issue Slot [inst/cycle]	1
Eligible Warps Per Scheduler [warp]	0.12	No Eligible [%]	94.49
Issued Warp Per Scheduler	0.06	One or More Eligible [%]	5.51

Warp State Statistics

Analysis of the states in which all warps spent cycles during the kernel execution. The warp states describe a warp's readiness or inability to issue its next instruction. The warp cycles per instruction define the latency between two consecutive instructions. The higher the value, the more warp parallelism is required to hide this latency. For each warp state, the chart shows the average number of cycles spent in that state per issued instruction. Stalls are not always impacting the overall performance nor are they completely avoidable. Only focus on stall reasons if the schedulers fail to issue every cycle. When executing a kernel with mixed library and user code, these metrics show the combined values.

Warp Cycles Per Issued Instruction [cycle]	253.15	Avg. Active Threads Per Warp	32
Warp Cycles Per Issue Active [warp]	253.15	Avg. Not Predicated Off Threads Per Warp	32
Warp Cycles Per Executed Instruction [cycle]	253.61		

GPU Speed Of Light

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 utilization with respect to the theoretical maximum. High-level overview of the utilization for compute and memory resources of the GPU presented as a roofline chart.

SOL SM [%]	6.23	Duration [usecond]	572.19
SOL Memory [%]	89.75	Elapsed Cycles [cycle]	622968
SOL L1/TEX Cache [%]	15.70	SM Active Cycles [cycle]	618515.41
SOL L2 Cache [%]	69.01	SM Frequency [cycle/nsecond]	1.09
SOL DRAM [%]	89.75	DRAM Frequency [cycle/nsecond]	1.21

GPU Utilization

SM [%]	6.23
Memory [%]	89.75

SOL SM Breakdown

SOL SM: Inst Executed Pipe Lsu [%]	6.23
SOL SM: Issue Active [%]	5.47
SOL SM: Inst Executed [%]	5.46
SOL SM: Mio2rf Writeback Active [%]	4.68
SOL SM: Mio Inst Issued [%]	3.90
SOL SM: Pipe Fma Cycles Active [%]	3.12
SOL SM: Mio Pq Write Cycles Active [%]	3.12
SOL SM: Inst Executed Pipe Adu [%]	3.12
SOL SM: Mio Pq Read Cycles Active [%]	3.12
SOL SM: Pipe Fp64 Cycles Active [%]	1.56
SOL SM: Pipe Shared Cycles Active [%]	1.56
SOL SM: Pipe Alu Cycles Active [%]	1.56
SOL SM: Inst Executed Pipe Cbu Pred On Any [%]	1.56
SOL SM: Inst Executed Pipe Uniform [%]	0.78
SOL IDC: Request Cycles Active [%]	0
SOL SM: Inst Executed Pipe Xu [%]	0
SOL SM: Inst Executed Pipe Tex [%]	0
SOL SM: Inst Executed Pipe Ipa [%]	0
SOL SM: Inst Executed Pipe Fp16 [%]	0
SOL SM: Pipe Tensor Cycles Active [%]	0

SOL Memory Breakdown

SOL GPU: Dram Throughput [%]	89.75
SOL L2: T Sectors [%]	51.96
SOL L2: D Sectors [%]	36.63
SOL L2: T Sectors Fill Device [%]	34.65
SOL L2: T Tag Requests [%]	26.01
SOL L2: Xbar2lts Cycles Active [%]	21.68
SOL L2: Lts2xbar Cycles Active [%]	17.34
SOL L1: M L1tex2xbar Req Cycles Active [%]	15.59
SOL L1: M Xbar2l1tex Read Sectors [%]	12.47
SOL L1: Data Pipe Lsu Wavefronts [%]	12.35
SOL L1: Lsu Writeback Active [%]	7.79
SOL L1: Lsuin Requests [%]	6.23
SOL L1: Data Bank Reads [%]	3.12
SOL L1: Data Bank Writes [%]	3.12
SOL L1: F Wavefronts [%]	0.00
SOL L1: Texin Sm2tex Req Cycles Active [%]	0.00
SOL L2: D Sectors Fill System [%]	0
SOL L1: Data Pipe Tex Wavefronts [%]	0
SOL L2: D Atomic Input Cycles Active [%]	0
SOL L1: Tex Writeback Active [%]	0

Floating Point Operations Roofline

Performance [FLOP/s] (1 = 1e+12)

Arithmetic Intensity [FLOP/byte]

Recommendations

- Bottleneck** 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.
- Roofline Analysis** The ratio of peak float (fp32) to double (fp64) performance on this device is 2:1. The kernel achieved 0% of this device's fp32 peak performance and 2% of its fp64 peak performance.

Nsight Compute w/ Stream

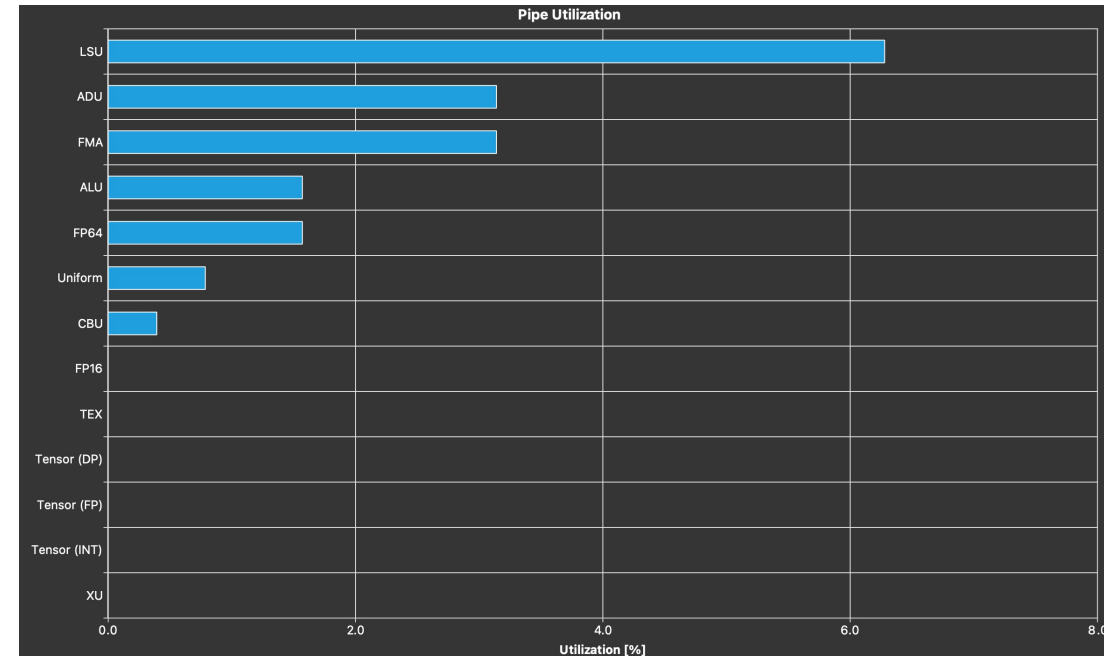
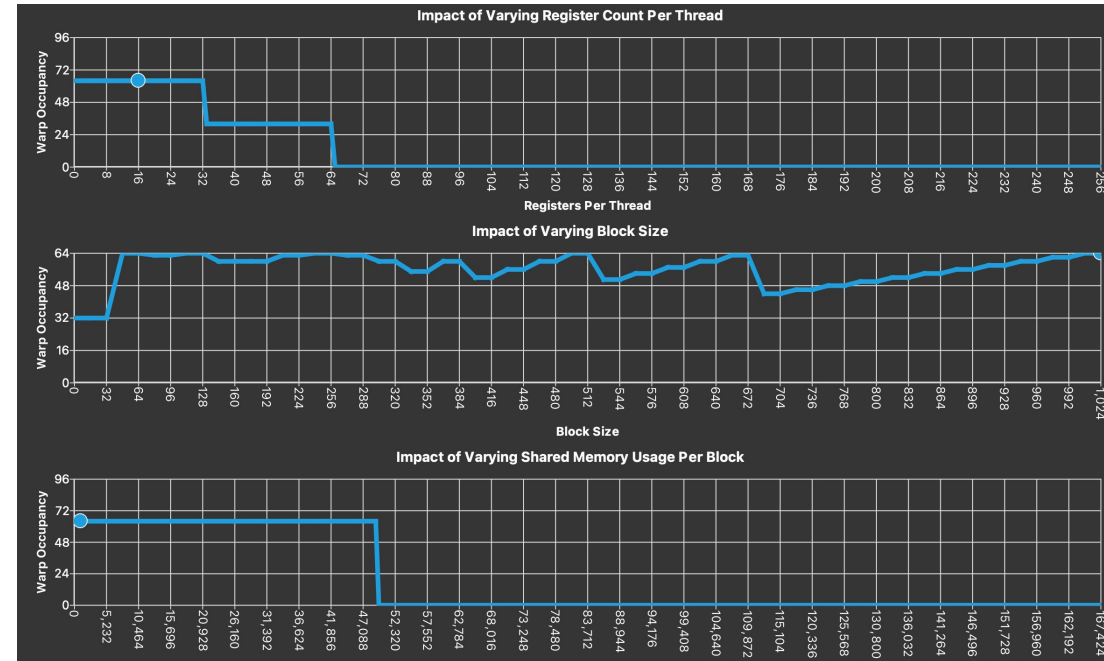
Page: Source Launch: 0 - 867 - triad_kernel Add Baseline Apply Rules Copy as Image

Current 867 - triad_ke... Time: 572.19 usecond Cycles: 622,968 Regs: 16 GPU: A100-SXM4-40GB SM Frequency: 1.09 cycle/insecond CC: 8.0 Process: [166971] cuda-stream

View: SASS

Source: triad_kernel Find... Navigation: Instructions Executed

#	Address	Source	Live Registers	Sampling Data (All)	Sampling Data (Not Issued)	Instructions Executed	Instructions Executed	Memory Address Space	Memory
triad_kernel									
1	00007f3b24f9c600	MOV R1, c[0x0][0x28]	1	52	37	1,048,576	33,554,432		
2	00007f3b24f9c610	S2R R8, SR_CTAID.X	2	36	10	1,048,576	33,554,432		
3	00007f3b24f9c620	MOV R9, 0x8	3	5	0	1,048,576	33,554,432		
4	00007f3b24f9c630	ULDC.64 UR4, c[0x0][0x118]	3	21	8	1,048,576	33,554,432		
5	00007f3b24f9c640	S2R R3, SR_TID.X	4	9	0	1,048,576	33,554,432		
6	00007f3b24f9c650	IMAD R8, R8, c[0x0][0x0], R3	4	127	33	1,048,576	33,554,432		
7	00007f3b24f9c660	IMAD.WIDE R2, R8, R9, c[0x0][0x168]	5	53	7	1,048,576	33,554,432		
8	00007f3b24f9c670	IMAD.WIDE R4, R8, R9, c[0x0][0x170]	7	186	41	1,048,576	33,554,432		
9	00007f3b24f9c680	LDG.E.64 R2, [R2.64]	7	66	18	1,048,576	33,554,432	Global	
10	00007f3b24f9c690	LDG.E.64 R4, [R4.64]	7	69	37	1,048,576	33,554,432	Global	
11	00007f3b24f9c6a0	IMAD.WIDE R8, R8, R9, c[0x0][0x160]	7	6	0	1,048,576	33,554,432		
12	00007f3b24f9c6b0	DFMA R6, R4, c[0x2][0x0], R2	9	15,400	14,674	1,048,576	33,554,432		
13	00007f3b24f9c6c0	STG.E.64 [R8.64], R6	5	35	28	1,048,576	33,554,432	Global	
14	00007f3b24f9c6d0	EXIT	1	6	0	1,048,576	33,554,432		
15	00007f3b24f9c6e0	BRA 0x7f3b24f9c6e0	0	248	215				
16	00007f3b24f9c6f0	NOP	0	0	0				
17	00007f3b24f9c700	NOP	0	0	0				
18	00007f3b24f9c710	NOP	0	0	0				
19	00007f3b24f9c720	NOP	0	0	0				
20	00007f3b24f9c730	NOP	0	0	0				
21	00007f3b24f9c740	NOP	0	0	0				
22	00007f3b24f9c750	NOP	0	0	0				
23	00007f3b24f9c760	NOP	0	0	0				
24	00007f3b24f9c770	NOP	0	0	0				



Items for demo

- nsys results
 - Running in the terminal
 - Reviewing report on local client
 - Kernels
 - Shows in event view
 - Description in event view
 - CUDA Memory Operations
- ncu results
 - Running in the terminal
 - Reviewing report on local client
 - Baselines for comparison of multiple kernels
 - Arithmetic Intensity of add and triad kernels on roofline analysis
 - ADD/FMA instructions in SASS

Case 2: Geometric Series

GeoSeries Benchmark

- Computing geometric series (i.e., $S = 1+r+r^2+r^3+\dots+r^{nGeo}$)
 - Input parameters
 - n^2 : array size
 - $nGeo$: the order of geometric series
 - Create a n^2 array (i.e., GeoR) with ratios: initialized by positive values less than 1.0
 - Repeat 100 times computation of geometric series (a n^2 array, GeoResult) with the given $nGeo$
- Build for A100
 - Using *nvhpc* module for OpenMP Target offloading
 - CFLAG for *nvc* compiler: *-mp=gpu -g -fast -O3*
- Comp_Geo kernel

```
#pragma omp target teams distribute parallel for collapse(2)
for(j=0;j<n;j++){ for(i=0;i<n;i++){
    id = i+j*n;
    tmpR = GeoR[id];          tmpResult = 1.0E0;
    for (iGeo=1;iGeo<=nGeo;iGeo++){
        tmpResult = 1.0E0 + tmpR*tmpResult; }
    GeoResult[id] = tmpResult;
}}
```

Offloading to GPU

Read one variable

$nGeo$ times FMA instructions

Write one variable

Theoretical Arithmetic Intensity for DP = $2*nGeo/(2*8)$

Performance of the benchmark on A100

Array size	Precision	nGEO	Wall time	FLOPs	Efficiency
8192 ²	DP	10	0.0088 s	1.52 TF/s	15.7%
8192 ²	DP	100	0.0201 s	6.67 TF/s	68.8%
8192 ²	DP	1000	0.1769 s	7.59 TF/s	78.2%
8192 ²	DP	10000	1.3921 s	9.64 TF/s	99.4%
8192 ²	SP	10	0.0072 s	1.86 TF/s	9.5%
8192 ²	SP	100	0.0128 s	10.46 TF/s	53.6%
8192 ²	SP	1000	0.0944 s	14.22 TF/s	72.9%
8192 ²	SP	10000	0.7444 s	18.03 TF/s	92.5%

Nsight Systems with GeoSeries benchmark

```
$ nsys profile -o out_nsys_DP_8192_10000 --stats=true --force-overwrite true ./Comp_GeoSeries_omp_nvc_DP 8192 10000
```

...

CUDA API Statistics:

Time(%)	Total Time (ns)	Num Calls	Average	Minimum	Maximum	StdDev	Name
96.1	1,571,893,187	13	120,914,860.5	7,093	178,287,856	54,664,365.4	cuStreamSynchronize
1.3	20,841,519	94	221,718.3	2,455	883,983	309,913.2	cuEventSynchronize
1.3	20,608,269	1	20,608,269.0	20,608,269	20,608,269	0.0	cuMemAllocManaged
1.1	17,893,965	1	17,893,965.0	17,893,965	17,893,965	0.0	cuMemHostAlloc

...

CUDA Kernel Statistics:

Time(%)	Total Time (ns)	Instances	Average	Minimum	Maximum	StdDev	Name
100.0	1,571,221,933	11	142,838,357.5	138,274,948	178,280,096	12,131,315.9	nvkernel_Comp_Geo_F1L30_1

CUDA Memory Operation Statistics (by time):

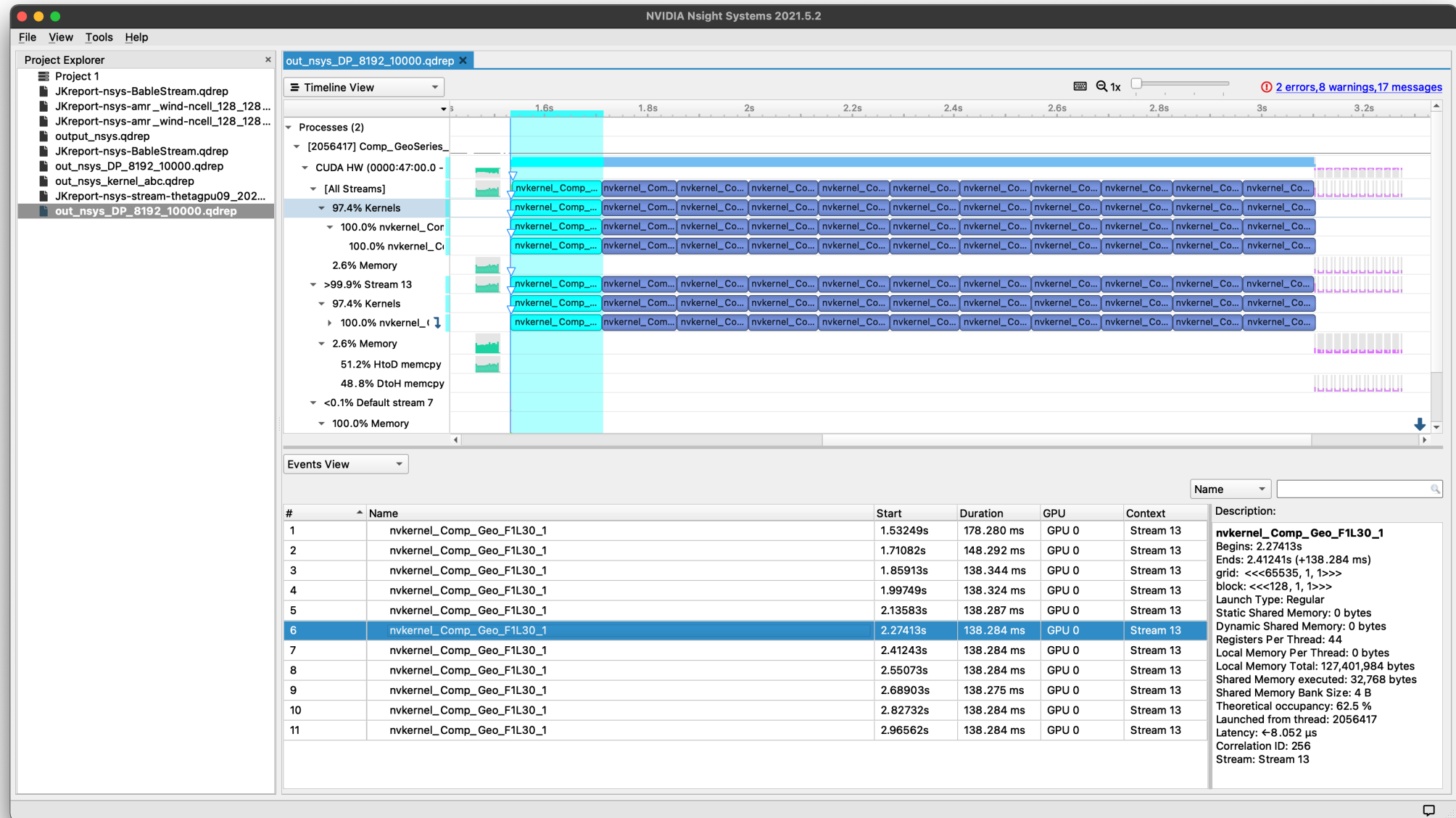
Time(%)	Total Time (ns)	Operations	Average	Minimum	Maximum	StdDev	Operation
51.2	21,839,952	33	661,816.7	3,072	954,008	133,998.1	[CUDA memcpy HtoD]
48.8	20,823,032	33	631,001.0	4,768	867,288	118,995.5	[CUDA memcpy DtoH]

CUDA Memory Operation Statistics (by size in KiB):

Total	Operations	Average	Minimum	Maximum	StdDev	Operation
524,288.000	33	15,887.515	0.008	16,384.000	2,852.087	[CUDA memcpy HtoD]
524,287.992	33	15,887.515	1.992	16,383.938	2,851.731	[CUDA memcpy DtoH]

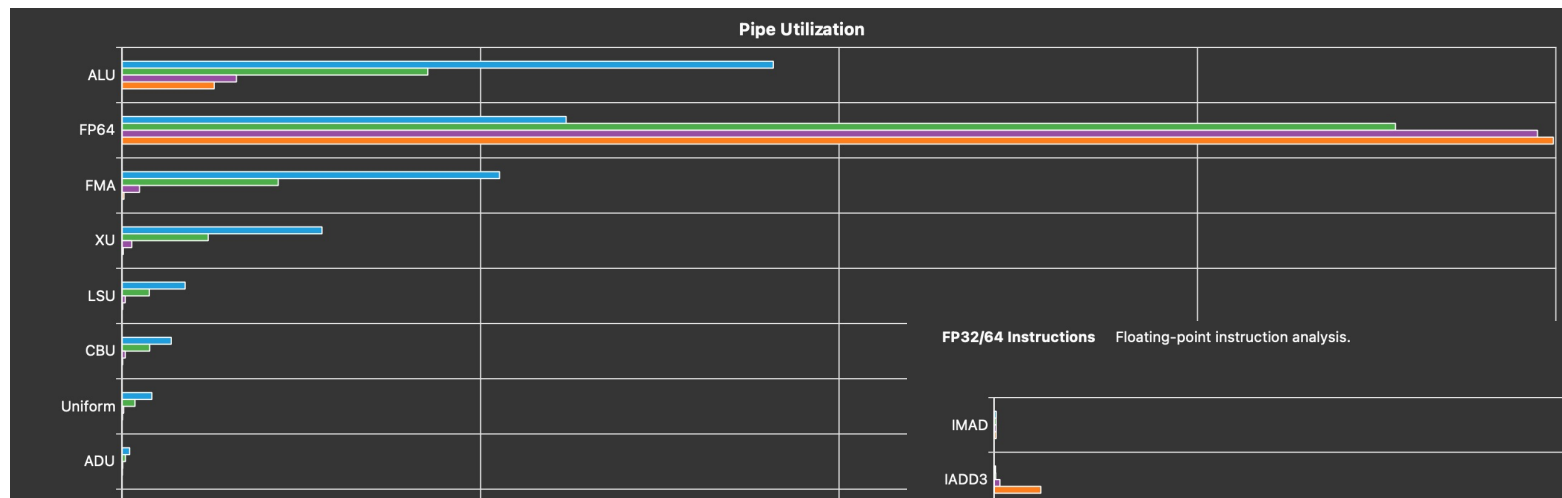
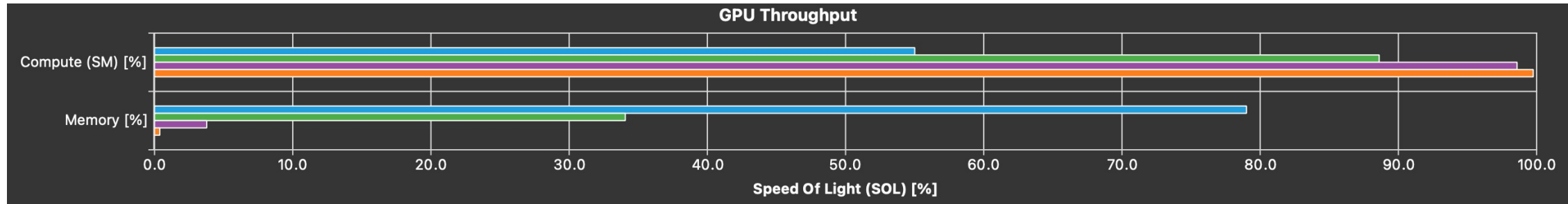
...

Nsight Systems with GeoSeries benchmark



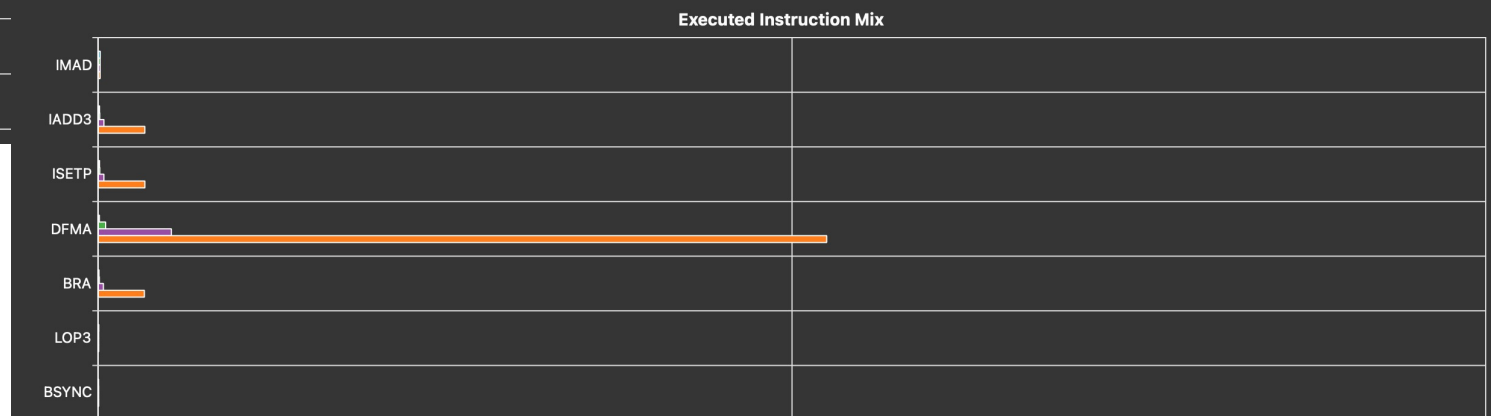
Nsight Compute w/ GeoSeries

nGEO=10 (blue)
 nGEO=100 (green)
 nGEO=1000 (purple)
 nGEO=10000 (orange)



ALU: Arithmetic Logic Unit
 FP64: Double-precision floating point unit
 FMA: FP32 FMA pipeline

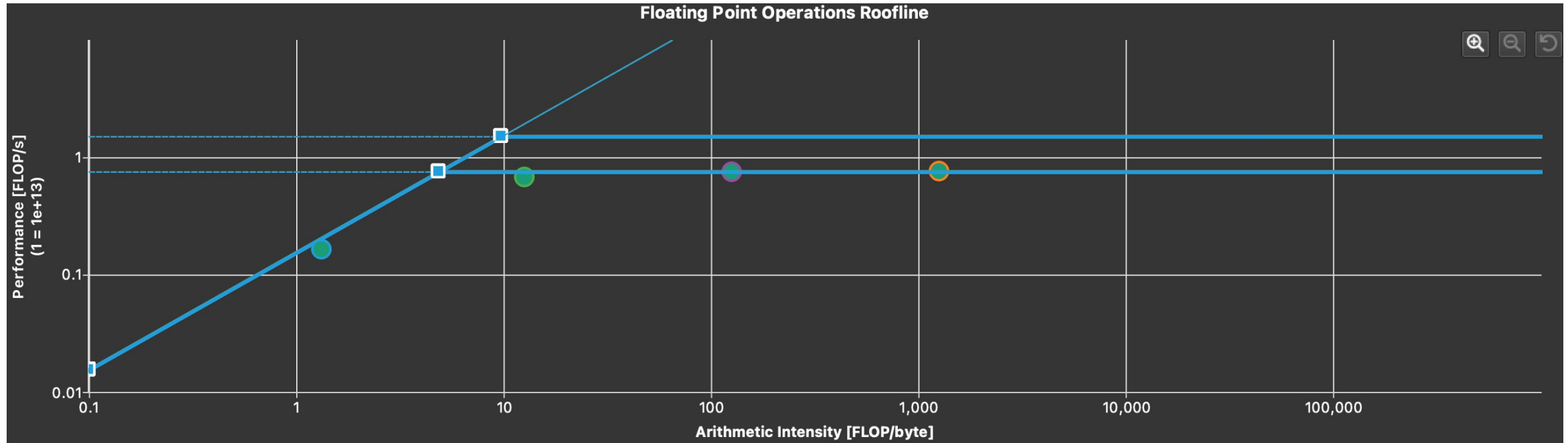
FP32/64 Instructions Floating-point instruction analysis.



DFMA: FP64 Fused Multiply Add

Nsight Compute w/ GeoSeries

nGEO=10 (blue)
nGEO=100 (green)
nGEO=1000 (purple)
nGEO=10000 (orange)



Precision	nGEO	Theoretical A/I	Measured A/I	Measured FLOPs
DP	10	1.25	1.33	1.63 TF/s
DP	100	12.5	12.68	6.71 TF/s
DP	1000	125	126.75	7.44 TF/s
DP	10000	1250	1266.92	7.53 TF/s

Items for demo

- nsys results
 - Reviewing report on local client
 - Kernels
 - Shows in event view
 - Description in event view
 - CUDA Memory Operations
- ncu results
 - Reviewing report on local client
 - Baselines for comparison of multiple executions
 - Details: GPU Throughput
 - Details: Pipe Utilization
 - Details: Executed Instruction Mix
 - Details: Floating Point Operation Roofline
 - Source: SASS
 - nGEO=10: DFMA (stall long scoreboard)
 - nGEO=10000: DFMA
 - FMA instructions in SASS

Thank you!