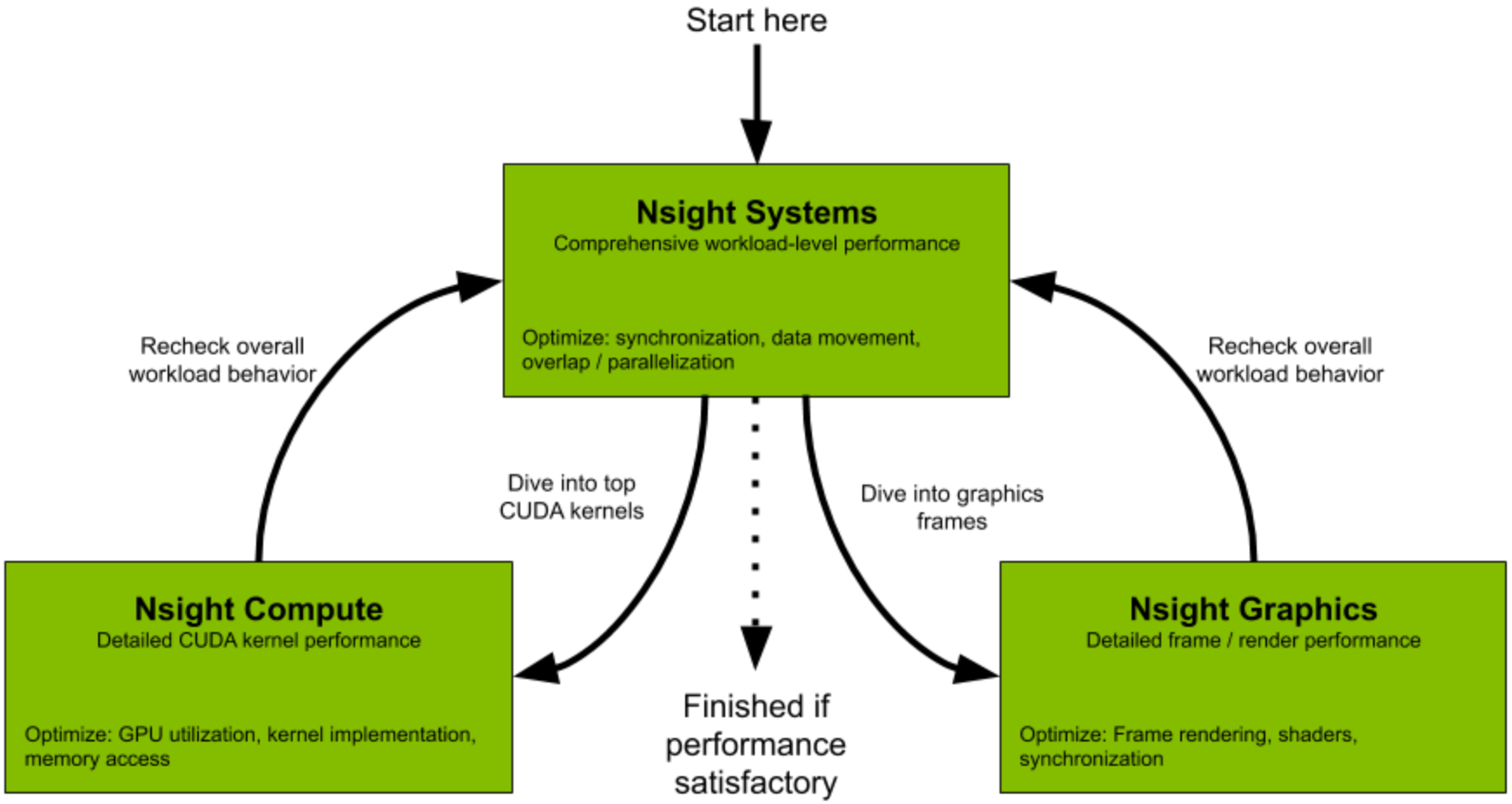

October 10-12, 2023
ALCF Hands-on HPC Workshop

Nsight Tools

Nsight Systems and Nsight Compute

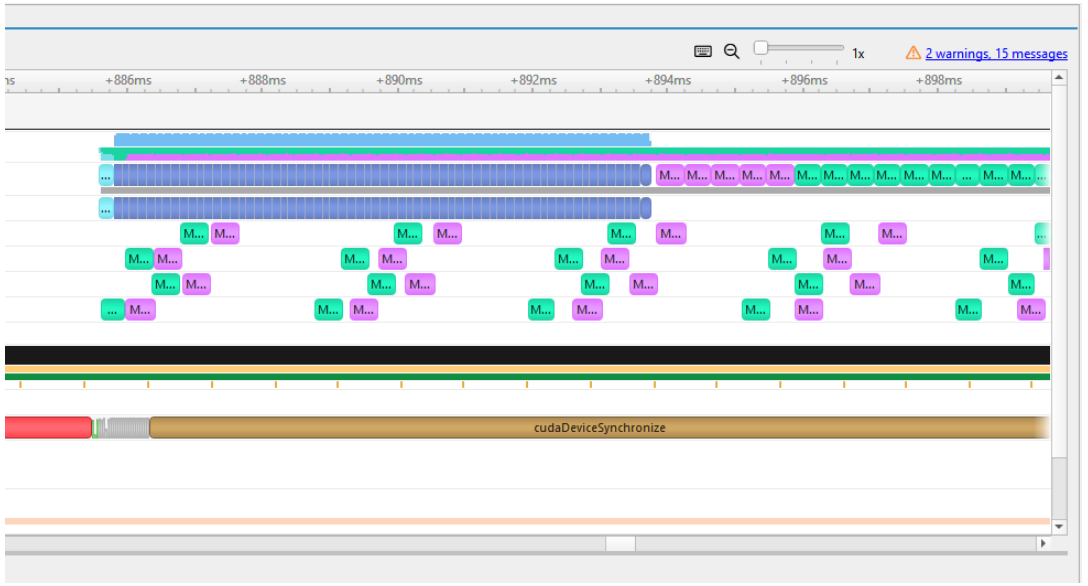
Matt Stack, NVIDIA

Overview



NSIGHT SYSTEMS

SYSTEM PROFILER

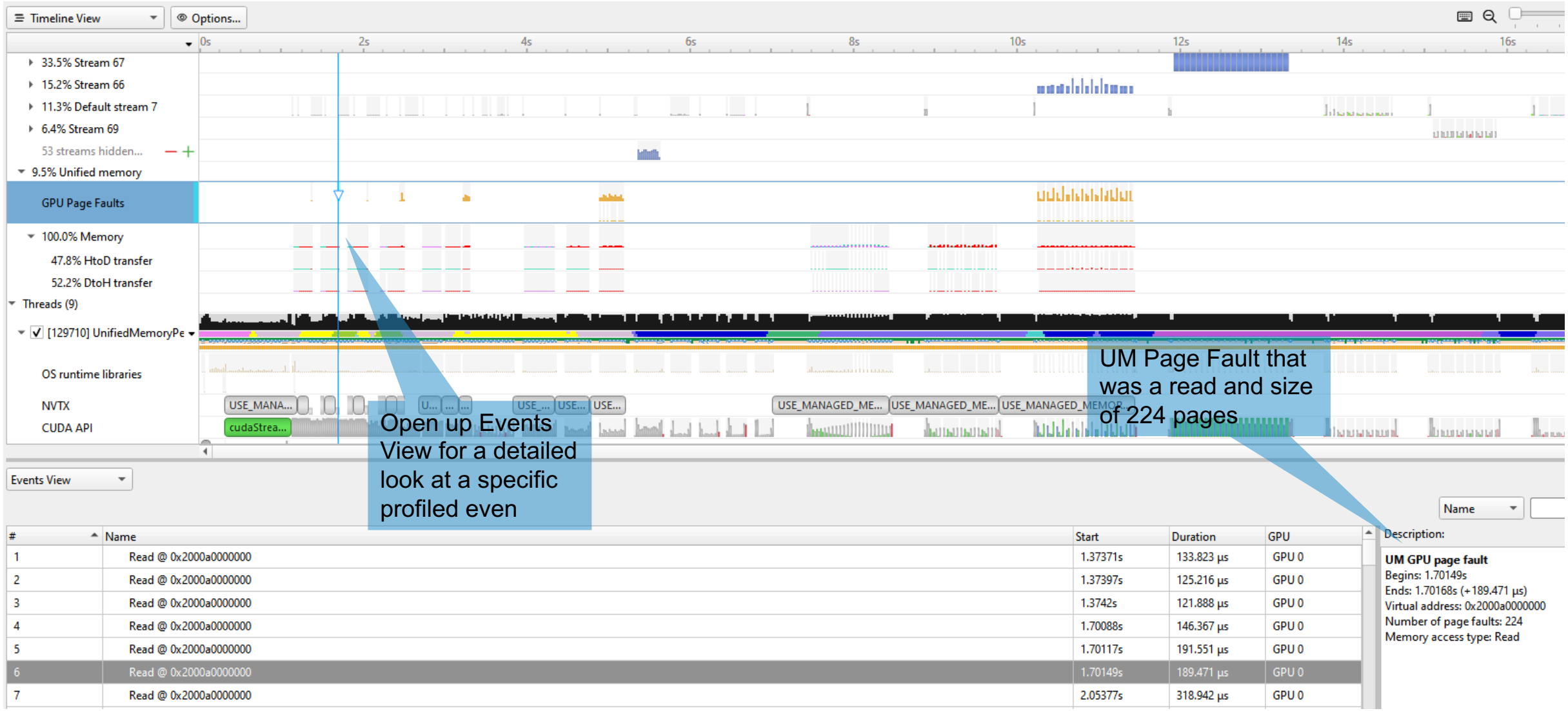


- System-wide view of application
  - Single timeline with GPU, CPU information
  - MPI, Memory patterns, multi-GPU
- Identify spots for performance tuning
  - Look for GPU idling, excessive synchronization, suboptimal memory traffic

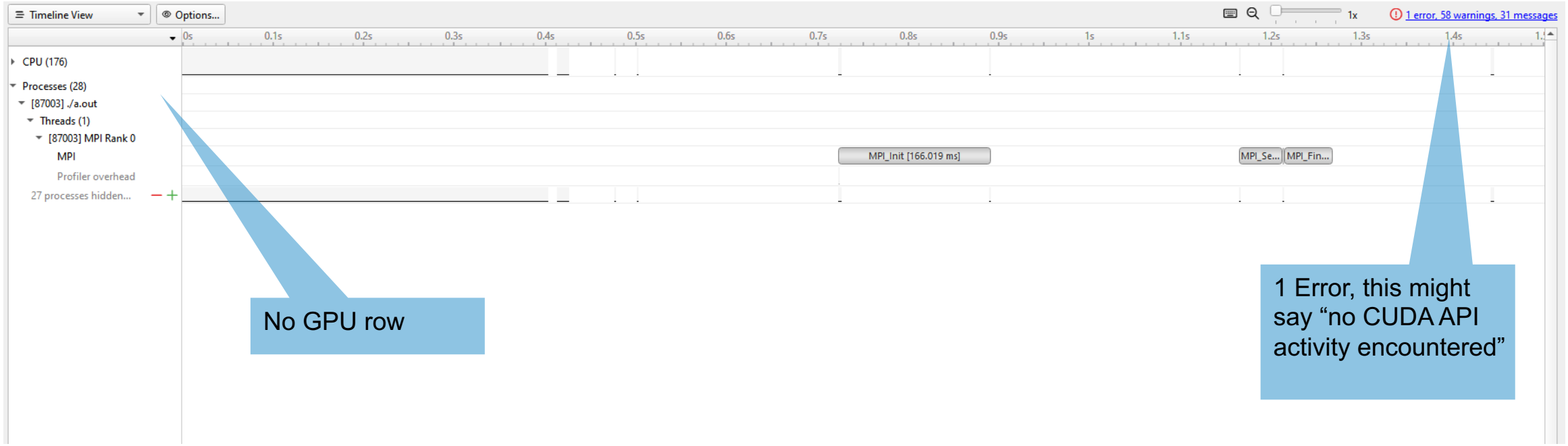
- ALCF Nsight docs: https://docs.alcf.anl.gov/theta-gpu/performance-tools/nvidia-nsight/
- Nsight Systems User guide from Nvidia: https://docs.nvidia.com/nsight-systems/UserGuide/index.html

- Available on Linux (x86, Power9, ARM), Windows, and Mac (Host only)

Time ->



- Trial and error, your first profile might look like this.. What went wrong?



How to run Nsight Systems on Polaris

- ALCF Beginners Guide: https://github.com/argonne-lcf/ALCFBeginnersGuide/blob/master/polaris/02_profiling.md

- Generally, Nsight Systems command looks like

  nsys profile -o <profile_name> --stats true ./<app exe>

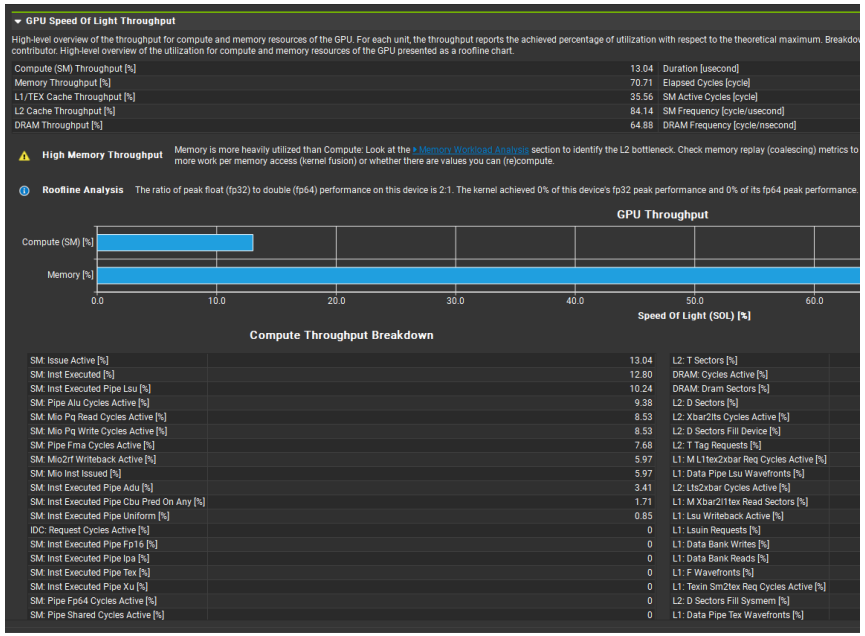
(+ any additional flags, here are two useful ones:

1) If using OpenACC or OpenMP, consider --trace openacc,openmp,cuda,nvtx,osrt. By default nsys traces API calls from CUDA, NVTX, opengl, and osrt),
2) If using UM, consider adding --cuda-um-gpu-page-faults true and --cude-um-cpu-page-faults true

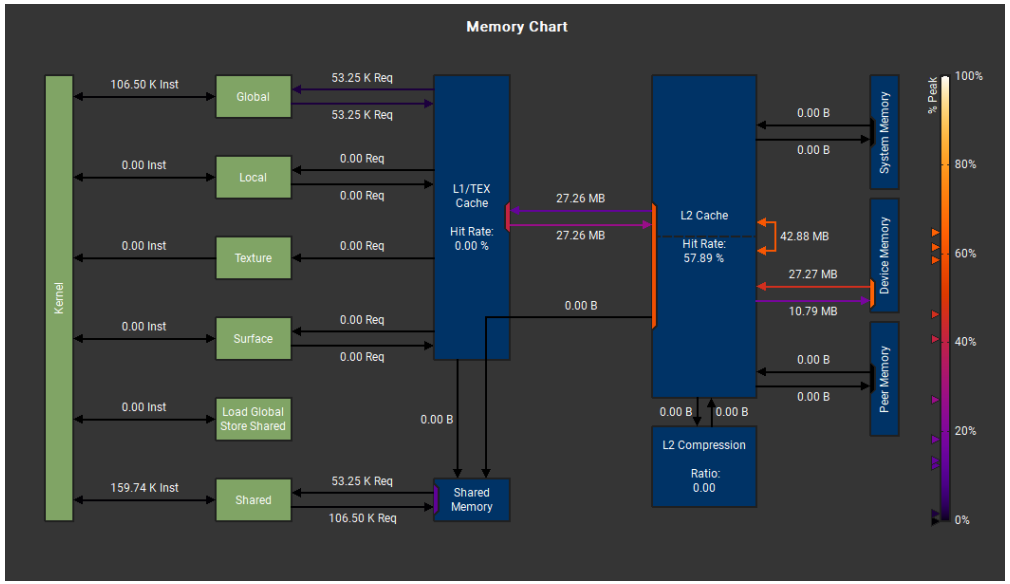
```
stack6@polaris-login-04:~> ml load nvhpc/23.3

The following have been reloaded with a version change:
  1) nvhpc/21.9 => nvhpc/23.3

stack6@polaris-login-04:~> which nsys
/opt/nvidia/hpc_sdk/Linux_x86_64/23.3/compilers/bin/nsys
stack6@polaris-login-04:~> nsys --version
NVIDIA Nsight Systems version 2023.1.1.127-32365746v0
```

```
stack6@polaris-login-04:~> ml load cudatoolkit-standalone/12.0.0
stack6@polaris-login-04:~> which nsys
/soft/compilers/cudatoolkit/cuda-12.0.0/bin/nsys
stack6@polaris-login-04:~> nsys --version
NVIDIA Nsight Systems version 2022.4.2.18-32044700v0
```

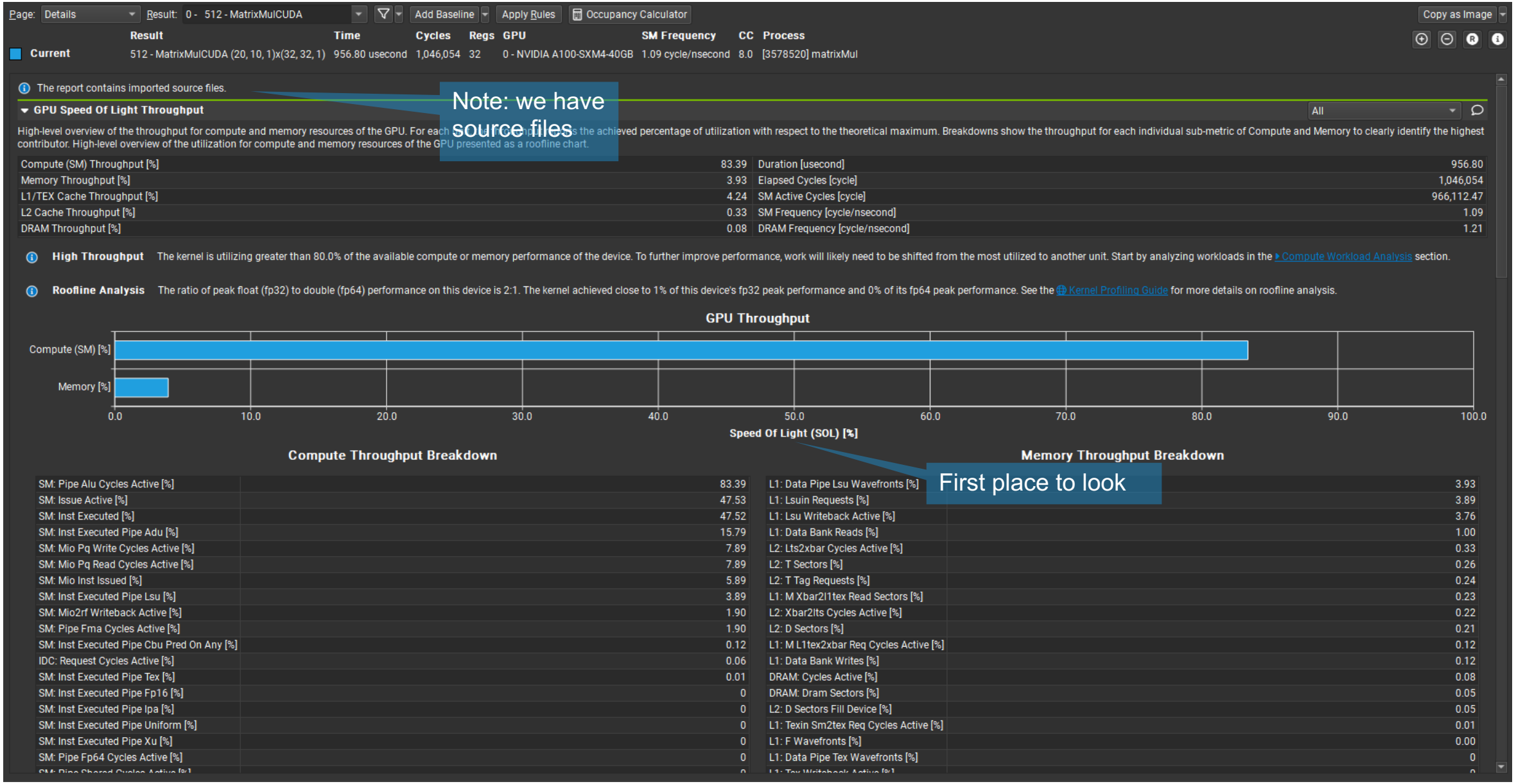
NSIGHT COMPUTE
KERNEL PROFILING TOOL

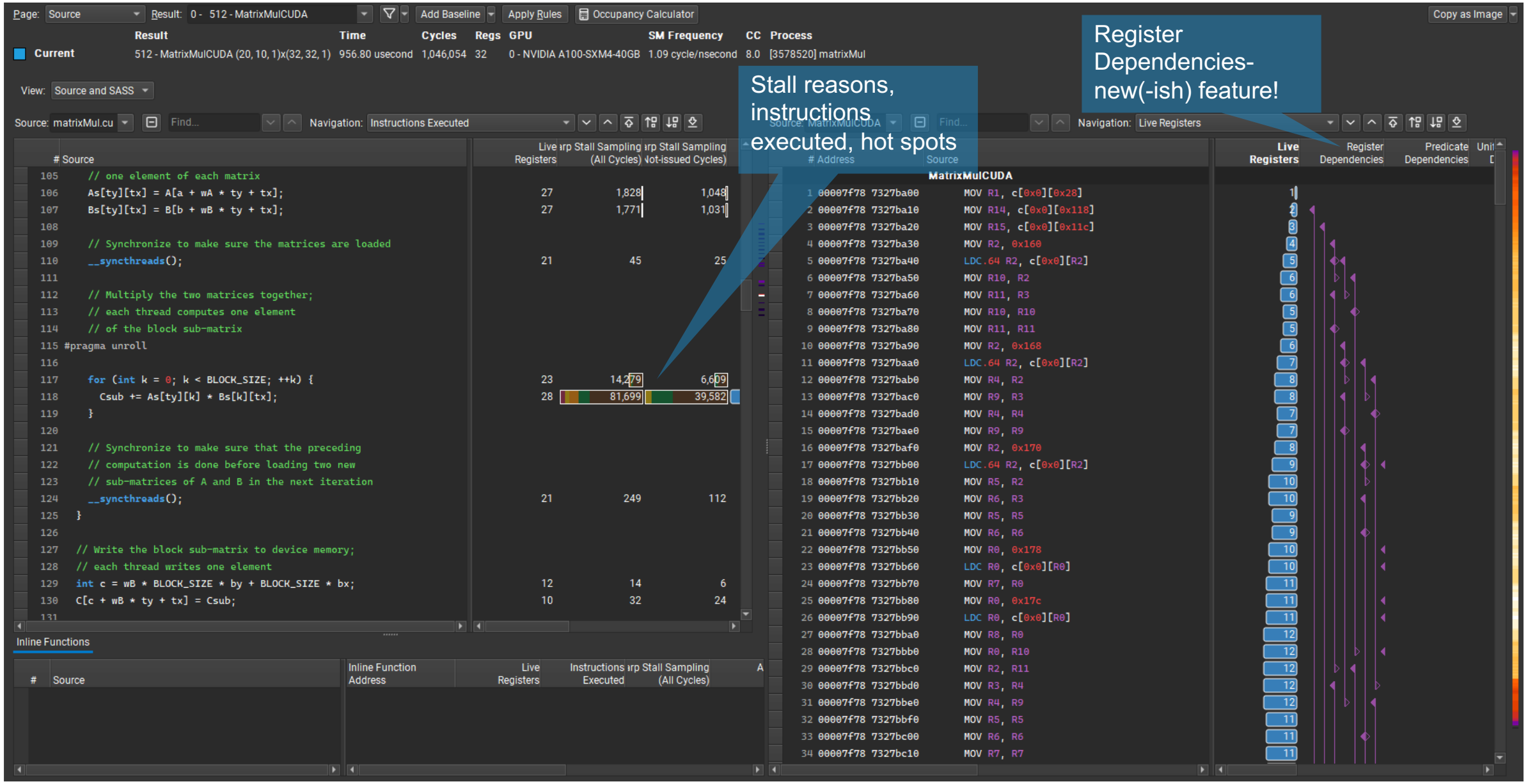


- "Zoomed in" profiling on a CUDA Kernel
  - Typically after Nsight Systems
- Compare multiple kernels with delta performance difference
  - "Did my code optimization make a change in the right direction?"
- View your kernel code with debugging information
  - Opt-in source code profiling

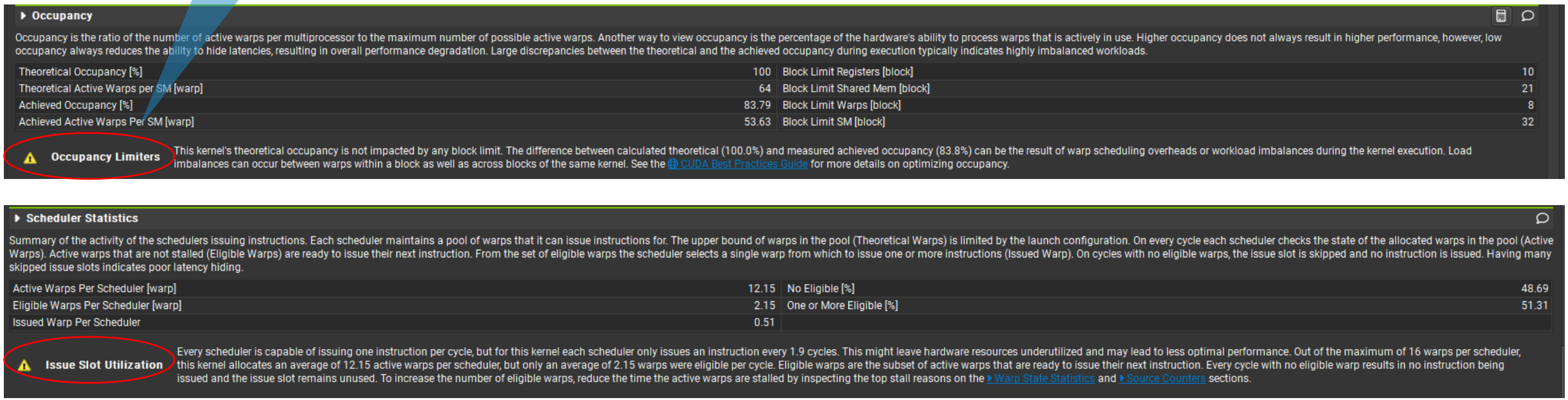


- Nsight Compute Docs: https://docs.nvidia.com/nsight-compute/NsightCompute/index.html
- Available on Linux (x86, Power9, ARM), Windows, and Mac (Host only)




Page: Source
Result: 0 - 512 - MatrixMulCUDA
Add Baseline
Apply Rules
Occupancy Calculator
Copy as Image
Result Time Cycles Regs GPU SM Frequency CC Process
Current 512 - MatrixMulCUDA (20, 10, 1)x(32, 32, 1) 956.80 usecond 1,046,054 32 0 - NVIDIA A100-SXM4-40GB 1.09 cycle/nsecond 8.0 [3578520] matrixMul
View: Source and SASS
Source: matrixMul.cu
Navigation: Instructions Executed
Navigation: Live Registers
Stall reasons, instructions executed, hot spots
Register Dependencies- new(-ish) feature!
Source
Live Registers
Warp Stall Sampling (All Cycles)
Warp Stall Sampling (Not-issued Cycles)
105 // one element of each matrix
106 As[ty][tx] = A[a + wA * ty + tx]; 27 1,828 1,048
107 Bs[ty][tx] = B[b + wB * ty + tx]; 27 1,771 1,031
108
109 // Synchronize to make sure the matrices are loaded
110 __syncthreads(); 21 45 25
111
112 // Multiply the two matrices together;
113 // each thread computes one element
114 // of the block sub-matrix
115 #pragma unroll
116
117 for (int k = 0; k < BLOCK_SIZE; ++k) { 23 14,279 6,609
118 Csub += As[ty][k] * Bs[k][tx]; 28 81,699 39,582
119 }
120
121 // Synchronize to make sure that the preceding
122 // computation is done before loading two new
123 // sub-matrices of A and B in the next iteration
124 __syncthreads(); 21 249 112
125 }
126
127 // Write the block sub-matrix to device memory;
128 // each thread writes one element
129 int c = wB * BLOCK_SIZE * by + BLOCK_SIZE * bx; 12 14 6
130 C[c + wB * ty + tx] = Csub; 10 32 24
131
Inline Functions
Source
Inline Function Address
Live Registers
Instructions Executed
Warp Stall Sampling (All Cycles)
Address
Source
MatrixMulCUDA
1 00007f78 7327ba00 MOV R1, c[0x0][0x28]
2 00007f78 7327ba10 MOV R14, c[0x0][0x118]
3 00007f78 7327ba20 MOV R15, c[0x0][0x11c]
4 00007f78 7327ba30 MOV R2, 0x160
5 00007f78 7327ba40 LDC.64 R2, c[0x0][R2]
6 00007f78 7327ba50 MOV R10, R2
7 00007f78 7327ba60 MOV R11, R3
8 00007f78 7327ba70 MOV R10, R10
9 00007f78 7327ba80 MOV R11, R11
10 00007f78 7327ba90 MOV R2, 0x168
11 00007f78 7327baa0 LDC.64 R2, c[0x0][R2]
12 00007f78 7327bab0 MOV R4, R2
13 00007f78 7327bac0 MOV R9, R3
14 00007f78 7327bad0 MOV R4, R4
15 00007f78 7327bae0 MOV R9, R9
16 00007f78 7327baf0 MOV R2, 0x170
17 00007f78 7327bb00 LDC.64 R2, c[0x0][R2]
18 00007f78 7327bb10 MOV R5, R2
19 00007f78 7327bb20 MOV R6, R3
20 00007f78 7327bb30 MOV R5, R5
21 00007f78 7327bb40 MOV R6, R6
22 00007f78 7327bb50 MOV R0, 0x178
23 00007f78 7327bb60 LDC R0, c[0x0][R0]
24 00007f78 7327bb70 MOV R7, R0
25 00007f78 7327bb80 MOV R0, 0x17c
26 00007f78 7327bb90 LDC R0, c[0x0][R0]
27 00007f78 7327bba0 MOV R8, R0
28 00007f78 7327bbb0 MOV R0, R10
29 00007f78 7327bbc0 MOV R2, R11
30 00007f78 7327bbd0 MOV R3, R4
31 00007f78 7327bbe0 MOV R4, R9
32 00007f78 7327bbf0 MOV R5, R5
33 00007f78 7327bc00 MOV R6, R6
34 00007f78 7327bc10 MOV R7, R7
Live Registers
Register Dependencies
Predicate Dependencies

Don’t ignore these Expert Analysis messages



Occupancy

Occupancy is the ratio of the number of active warps per multiprocessor to the maximum number of possible active warps. Another way to view occupancy is the percentage of the hardware's ability to process warps that is actively in use. Higher occupancy does not always result in higher performance, however, low occupancy always reduces the ability to hide latencies, resulting in overall performance degradation. Large discrepancies between the theoretical and the achieved occupancy during execution typically indicates highly imbalanced workloads.

Theoretical Occupancy [%]	100	Block Limit Registers [block]	10
Theoretical Active Warps per SM [warp]	64	Block Limit Shared Mem [block]	21
Achieved Occupancy [%]	83.79	Block Limit Warps [block]	8
Achieved Active Warps Per SM [warp]	53.63	Block Limit SM [block]	32

Occupancy Limiters This kernel's theoretical occupancy is not impacted by any block limit. The difference between calculated theoretical (100.0%) and measured achieved occupancy (83.8%) can be the result of warp scheduling overheads or workload imbalances during the kernel execution. Load imbalances can occur between warps within a block as well as across blocks of the same kernel. See the CUDA Best Practices Guide for more details on optimizing occupancy.

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]	12.15	No Eligible [%]	48.69
Eligible Warps Per Scheduler [warp]	2.15	One or More Eligible [%]	51.31
Issued Warp Per Scheduler	0.51		

Issue Slot Utilization Every scheduler is capable of issuing one instruction per cycle, but for this kernel each scheduler only issues an instruction every 1.9 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 12.15 active warps per scheduler, but only an average of 2.15 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, reduce the time the active warps are stalled by inspecting the top stall reasons on the Warp State Statistics and Source Counters sections.

How to run Nsight Compute on Polaris

- ALCF Beginners Guide: https://github.com/argonne-lcf/ALCFBeginnersGuide/blob/master/polaris/02_profiling.md

- Generally, Nsight Compute command looks like

  ncu -o <profile_name> -k <kernel_name> -c 1 ./<app exe>

(+ the addition of any additional flags, here are two useful ones:

1) --import-source true (IF nvcc had the flag -lineinfo)
2) --set detailed or --set full to get Memory Analysis section and Roofline, default is --set basic

Compute Sanitizer

- Correctness checking tool
- Drop-in replacement for depreciated cuda-memcheck

Memcheck	Racecheck	Initcheck	Synccheck
Out-of-bounds and misaligned memory access	Race conditions for *shared* memory	Uninitialized device global memory detector	Thread synchronization hazard detector

Docs: https://docs.nvidia.com/compute-sanitizer/ComputeSanitizer/index.html

Compute-sanitizer --tool <which tool> ./app <app options>

Resources

- Nsight Compute memory analysis deep dive: https://www.nvidia.com/en-us/on-demand/session/gtcspring21-s32089/
- Nsight Systems user guide: https://docs.nvidia.com/nsight-systems/UserGuide/index.html
- Nsight Compute CLI user guide: https://docs.nvidia.com/nsight-compute/NsightComputeCli/index.html
- Nsight Compute GUI user guide: https://docs.nvidia.com/nsight-compute/NsightCompute/index.html
- Analysis Driven Optimization (ADO) with Nsight Compute Dev Blog: https://developer.nvidia.com/blog/analysis-driven-optimization-preparing-for-analysis-with-nvidia-nsight-compute-part-1/
- “What The Profiler Is Telling You” GTC talk: https://www.nvidia.com/en-us/on-demand/session/gtcsj20-s22141/
- Roofline analysis- ALCF 2021: https://www.youtube.com/watch?v=fsC3QeZHM1U