INSTRUCTION MIX SAMPLE

v2023.1.1 | March 2024

TABLE OF CONTENTS

Chapter 1.
INTRODUCTION

This sample profiles a CUDA kernel which applies a simple sobel edge detection filter to an image in global memory using the Nsight Compute profiler. The profiler is used to analyze and identify the performance bottleneck due to an imbalanced instruction mix.

Chapter 2. APPLICATION

This sample CUDA application applies a simple sobel edge detection filter to an image in global memory. The input and output images are at separate memory locations. For simplicity it only handles image sizes which are an integral multiple of block size. (**`BLOCK_SIZE`** - defined in the source file **`"instructionMix.cu"`**)

The instructionMix sample is available with Nsight Compute under **`<nsight-compute-install-directory>/extras/samples/instructionMix`**.

Chapter 3. CONFIGURATION

The profiling results included in this document were collected on the following configuration:

- Target system: Linux (x86_64) with a NVIDIA RTX A4500 (Ampere GA102) GPU
- Nsight Compute version: 2023.3.1

The Nsight Compute UI screen shots in the document are taken by opening the profiling reports on a Windows 10 system.

Chapter 4.
INITIAL VERSION OF THE KERNEL

The Sobel operator performs a 2-D spatial gradient measurement on an image which emphasizes regions of high spatial frequency that correspond to edges. Typically it is used to find the approximate absolute gradient magnitude at each point in an input grayscale image. Each thread applies the Sobel operator to one pixel of the input image and generates one pixel of the output image. The operator uses two 3x3 kernels which are convolved with the original image to calculate approximations of the derivaties - one for horizontal changes, and one for vertical. The **`Sobel`** kernel is defined as a function template that can be used as a generic function for different floating point precisions.

```
template<typename FLOAT_T>
__global__ void Sobel(
    uchar4* pOut,
    uchar4* pImg,
    const int imgWidth,
    const int imgHeight)
{
   const int tx = blockIdx.x * blockDim.x + threadIdx.x;
   const int ty = blockIdx.y * blockDim.y + threadIdx.y;
   const int outIdx = ty * imgWidth + tx;

   const int SX[] = {1, 2, 1, 0, 0, 0, -1, -2, -1};
   const int SY[] = {1, 0, -1, 2, 0, -2, 1, 0, -1};

   FLOAT_T sumX = 0.;
   FLOAT_T sumY = 0.;
   for (int j = -1; j <= 1; ++j)
   {
       for (int i = -1; i <= 1; ++i)
       {
           const auto idx = (j + 1) * 3 + (i + 1);
           const auto sx = SX[idx];
           const auto sy = SY[idx];

           const auto luminance = GetPixel(pImg, tx + i, ty + j, imgWidth,
 imgHeight);
           sumX += (FLOAT_T)luminance * (FLOAT_T)sx;
           sumY += (FLOAT_T)luminance * (FLOAT_T)sy;
       }
   }

   sumX /= (FLOAT_T)9.;
   sumY /= (FLOAT_T)9.;

   const FLOAT_T threshold = 24.;
   if (sumX > threshold || sumY > threshold)
   {
       pOut[outIdx] = make_uchar4(0, 255, 255, 0);
   }

}
```

The initial version of the kernel **`Sobel`** executes the math operations on the grayscale values in double precision floating point accuracy.

```
       Sobel<double><<<grid, block>>>( pDstImage, pSrcImage, imgWidth,
 imgHeight);
```

Profile the initial version of the kernel

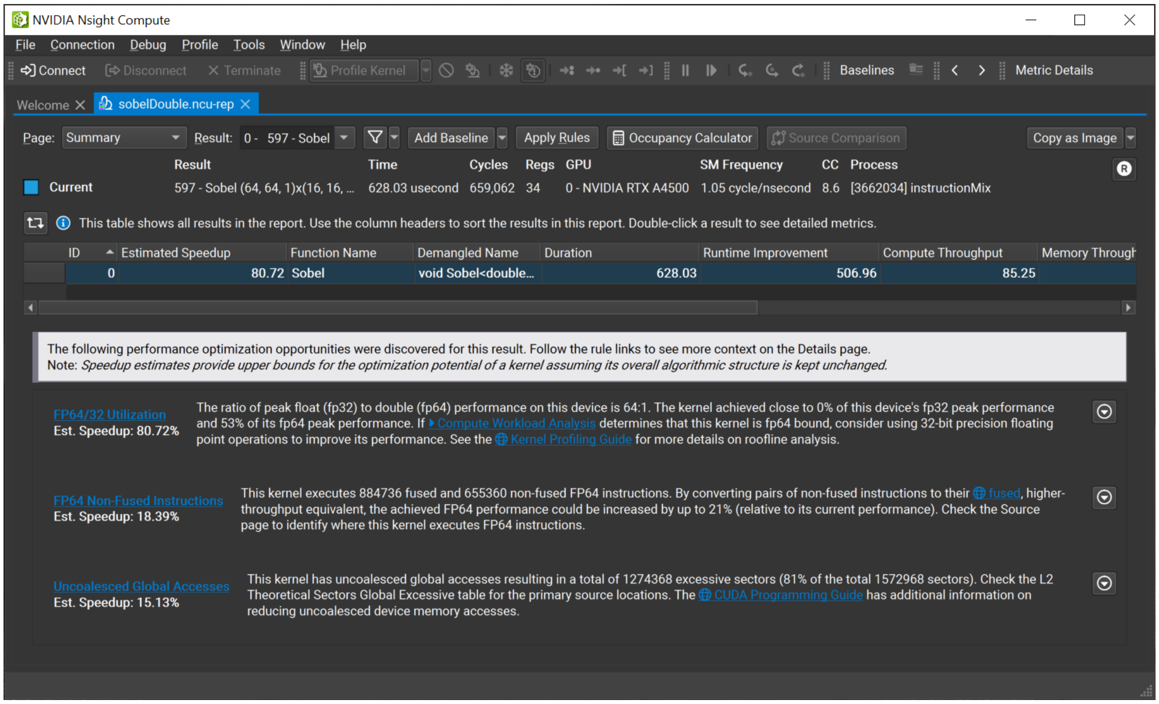
There are multiple ways to profile kernels with Nsight Compute. For full details see the Nsight Compute Documentation. One example is to perform the following steps:

- Refer to the **`README`** distributed with the sample on how to build the application
- Run **`ncu-ui`** on the host system
- Use a local connection if the GPU is on the host system. If the GPU is on a remote system, set up a remote connection to the target system
- Use the **`Profile`** activity to profile the sample application

- Choose the **`full`** section set
- Use defaults for all other options
- Set a report name and then click on **`Launch`**

Summary page

The **`Summary`** page lists the kernels profiled and provides some key metrics for each profiled kernel. It also lists the performance opportunities and estimated speedup for each.



	Result	Time	Cycles	Regs	GPU	SM Frequency	CC	Process
Current	597 - Sobel (64, 64, 1)x(16, 16, ...	628.03 usecond	659,062	34	0 - NVIDIA RTX A4500	1.05 cycle/nsecond	8.6	[3662034] instructionMix

This table shows all results in the report. Use the column headers to sort the results in this report. Double-click a result to see detailed metrics.

ID	Estimated Speedup	Function Name	Demangled Name	Duration	Runtime Improvement	Compute Throughput	Memory Throug
0	80.72	Sobel	void Sobel<double...	628.03	506.96	85.25	

The following performance optimization opportunities were discovered for this result. Follow the rule links to see more context on the Details page.
Note: *Speedup estimates provide upper bounds for the optimization potential of a kernel assuming its overall algorithmic structure is kept unchanged.*

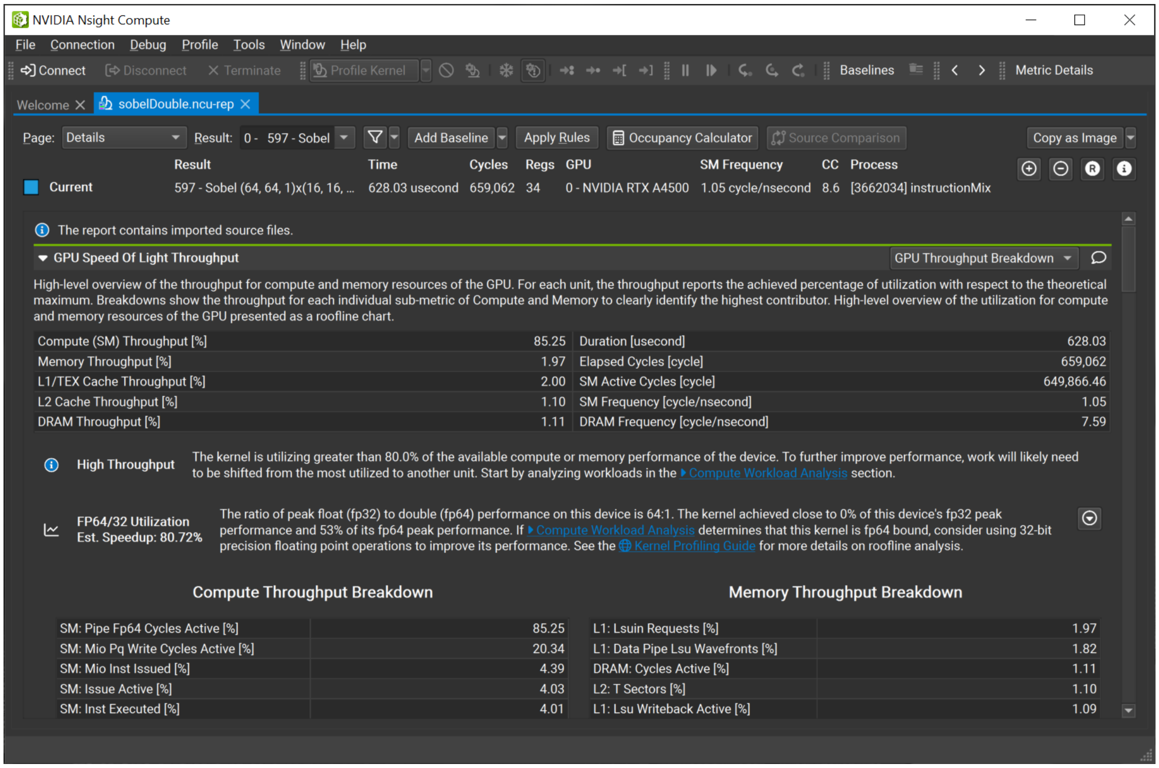
FP64/32 Utilization Est. Speedup: 80.72%	The ratio of peak float (fp32) to double (fp64) performance on this device is 64:1. The kernel achieved close to 0% of this device's fp32 peak performance and 53% of its fp64 peak performance. If Compute Workload Analysis determines that this kernel is fp64 bound, consider using 32-bit precision floating point operations to improve its performance. See the Kernel Profiling Guide for more details on roofline analysis.
FP64 Non-Fused Instructions Est. Speedup: 18.39%	This kernel executes 884736 fused and 655360 non-fused FP64 instructions. By converting pairs of non-fused instructions to their fused, higher-throughput equivalent, the achieved FP64 performance could be increased by up to 21% (relative to its current performance). Check the Source page to identify where this kernel executes FP64 instructions.
Uncoalesced Global Accesses Est. Speedup: 15.13%	This kernel has uncoalesced global accesses resulting in a total of 1274368 excessive sectors (81% of the total 1572968 sectors). Check the L2 Theoretical Sectors Global Excessive table for the primary source locations. The CUDA Programming Guide has additional information on reducing uncoalesced device memory accesses.

For this kernel it shows a hint for **`FP64/32 Utilization`** and suggests using 32-bit precision floating point operations to improve performance. Click on **`FP64/32 Utilization`** rule link to see more context on the **`Details`** page. It opens the **`GPU Speed of Light Throughput`** section on the **`Details`** page.

Details page - GPU Speed Of Light Throughput

The **`Details`** page **`GPU Speed Of Light Throughput`** section provides a high-level overview of the throughput for compute and memory resources of the GPU used by the kernel.

The initial version of the kernel has a duration of 628.03 microseconds and this is used as the baseline for further optimizations.


NVIDIA Nsight Compute
File Connection Debug Profile Tools Window Help
Connect Disconnect Terminate Profile Kernel Baselines Metric Details
Welcome sobelDouble.ncu-rep
Add Baseline Apply Rules Occupancy Calculator Source Comparison Copy as Image
GPU Throughput Breakdown

Page: Details Result: 0 - 597 - Sobel

	Result	Time	Cycles	Regs	GPU	SM Frequency	CC	Process
Current	597 - Sobel (64, 64, 1)x(16, 16, ...	628.03 usecond	659,062	34	0 - NVIDIA RTX A4500	1.05 cycle/nsecond	8.6	[3662034] instructionMix

The report contains imported source files.

GPU Speed Of Light Throughput

High-level overview of the throughput for compute and memory resources of the GPU. For each unit, the throughput reports the achieved percentage of utilization with respect to the theoretical maximum. Breakdowns show the throughput for each individual sub-metric of Compute and Memory to clearly identify the highest contributor. High-level overview of the utilization for compute and memory resources of the GPU presented as a roofline chart.

Compute (SM) Throughput [%]	85.25	Duration [usecond]	628.03
Memory Throughput [%]	1.97	Elapsed Cycles [cycle]	659,062
L1/TEX Cache Throughput [%]	2.00	SM Active Cycles [cycle]	649,866.46
L2 Cache Throughput [%]	1.10	SM Frequency [cycle/nsecond]	1.05
DRAM Throughput [%]	1.11	DRAM Frequency [cycle/nsecond]	7.59

High Throughput: 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 Compute Workload Analysis section.

FP64/32 Utilization Est. Speedup: 80.72%: The ratio of peak float (fp32) to double (fp64) performance on this device is 64:1. The kernel achieved close to 0% of this device's fp32 peak performance and 53% of its fp64 peak performance. If Compute Workload Analysis determines that this kernel is fp64 bound, consider using 32-bit precision floating point operations to improve its performance. See the Kernel Profiling Guide for more details on roofline analysis.

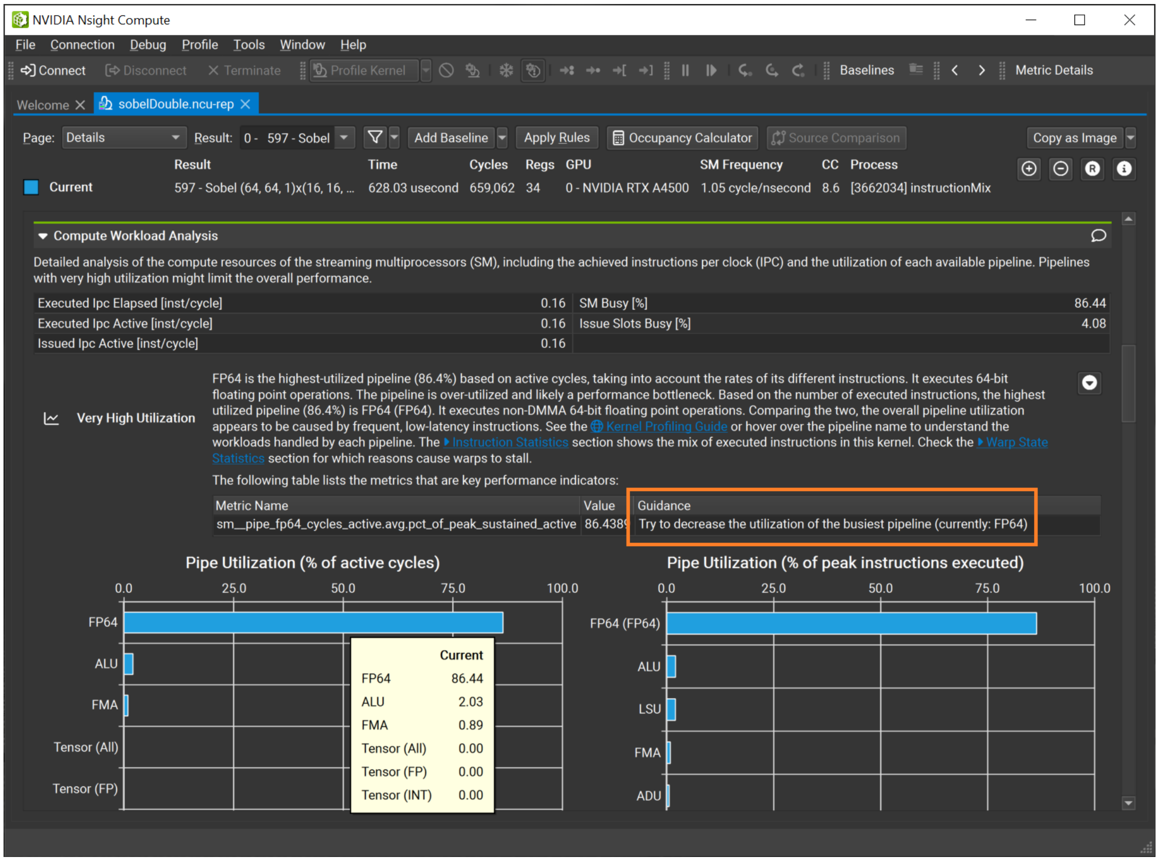
Compute Throughput Breakdown		Memory Throughput Breakdown	
SM: Pipe Fp64 Cycles Active [%]	85.25	L1: Lsuin Requests [%]	1.97
SM: Mio Pq Write Cycles Active [%]	20.34	L1: Data Pipe Lsu Wavefronts [%]	1.82
SM: Mio Inst Issued [%]	4.39	DRAM: Cycles Active [%]	1.11
SM: Issue Active [%]	4.03	L2: T Sectors [%]	1.10
SM: Inst Executed [%]	4.01	L1: Lsu Writeback Active [%]	1.09

For this kernel it shows a hint for **`High Throughput`** and **`FP64/32 Utilization`** and suggests looking at the **`Compute Workload Analysis`** section. Also we can see the **`GPU Throughput Breakdown`** tables at the bottom for Compute Throughput and Memory Throughput. The Compute Throughput Breakdown table shows that the SM FP64 pipe throughput is high (85.25%). Click on **`Compute Workload Analysis`** to analyze the usage of compute resources of the streaming multiprocessors (SM).

Details page - Compute Workload Analysis section

The **`Compute Workload Analysis`** section shows a hint for **`Very High Utilization`**. It shows that FP64 is the highest-utilized pipeline (86.44%). The FP64 pipeline executes 64-bit floating point operations. It mentions that the pipeline is over-utilized and likely a performance bottleneck. The guidance provided is to try and decrease the utlization of the FP64 pipeline.

Initial version of the kernel



	Result	Time	Cycles	Regs	GPU	SM Frequency	CC	Process
Current	597 - Sobel (64, 64, 1)x(16, 16, ...	628.03 usecond	659,062	34	0 - NVIDIA RTX A4500	1.05 cycle/nsecond	8.6	[3662034] instructionMix

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]	0.16	SM Busy [%]	86.44
Executed Ipc Active [inst/cycle]	0.16	Issue Slots Busy [%]	4.08
Issued Ipc Active [inst/cycle]	0.16		

Very High Utilization

FP64 is the highest-utilized pipeline (86.4%) based on active cycles, taking into account the rates of its different instructions. It executes 64-bit floating point operations. The pipeline is over-utilized and likely a performance bottleneck. Based on the number of executed instructions, the highest utilized pipeline (86.4%) is FP64 (FP64). It executes non-DMMA 64-bit floating point operations. Comparing the two, the overall pipeline utilization appears to be caused by frequent, low-latency instructions. See the Kernel Profiling Guide or hover over the pipeline name to understand the workloads handled by each pipeline. The Instruction Statistics section shows the mix of executed instructions in this kernel. Check the Warp State Statistics section for which reasons cause warps to stall.

The following table lists the metrics that are key performance indicators:

Metric Name	Value	Guidance
sm__pipe_fp64_cycles_active.avg.pct_of_peak_sustained_active	86.438	Try to decrease the utilization of the busiest pipeline (currently: FP64)

Pipe Utilization (% of active cycles)

	Current
FP64	86.44
ALU	2.03
FMA	0.89
Tensor (All)	0.00
Tensor (FP)	0.00
Tensor (INT)	0.00

Pipe Utilization (% of peak instructions executed)

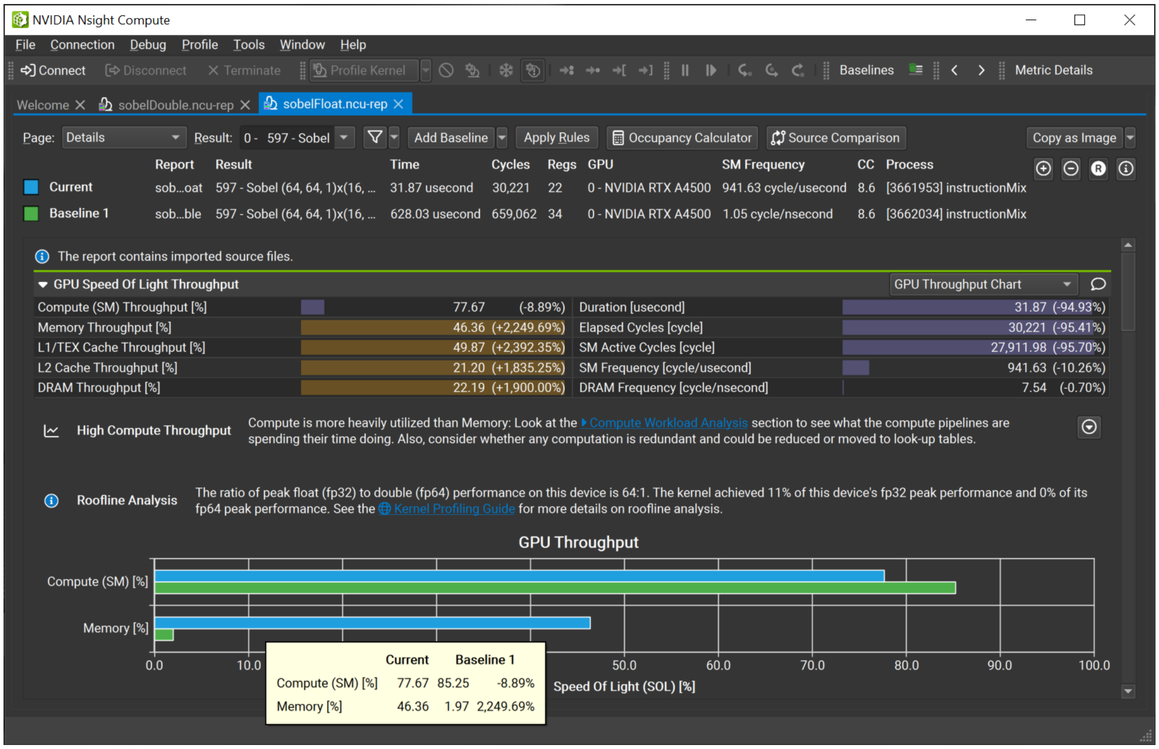
Chapter 5. UPDATED VERSION OF THE KERNEL

Based on the profiler hint of high FP64 pipeline utilization, we modify the code to use single precision floating point instead of double precision. Since our input image has a very limited value range and the Sobel operator is not receptible to minor differences in precision, switching the computations from double to single precision has no negative impact on its functionality.

```
      Sobel<float><<<grid, block>>>( pDstImage, pSrcImage, imgWidth,
imgHeight);
```

Profile the updated kernel

The kernel duration has reduced from 628.03 microseconds to 31.87 microseconds. We can set a baseline to the initial version of the kernel and compare the profiling results.


NVIDIA Nsight Compute
File Connection Debug Profile Tools Window Help
Connect Disconnect Terminate Profile Kernel Baselines Metric Details
Welcome sobelDouble.ncu-rep sobelFloat.ncu-rep
Page: Details Result: 0 - 597 - Sobel Add Baseline Apply Rules Occupancy Calculator Source Comparison Copy as Image
GPU Throughput Chart
GPU Throughput
Compute (SM) [%]
Memory [%]
0.0
10.0
50.0
60.0
70.0
80.0
90.0
100.0
Speed Of Light (SOL) [%]

	Report	Result	Time	Cycles	Regs	GPU	SM Frequency	CC	Process
🟦 Current	sob...oat	597 - Sobel (64, 64, 1)x(16, ...	31.87 usecond	30,221	22	0 - NVIDIA RTX A4500	941.63 cycle/usecond	8.6	[3661953] instructionMix
🟩 Baseline 1	sob...ble	597 - Sobel (64, 64, 1)x(16, ...	628.03 usecond	659,062	34	0 - NVIDIA RTX A4500	1.05 cycle/nsecond	8.6	[3662034] instructionMix

The report contains imported source files.

GPU Speed Of Light Throughput

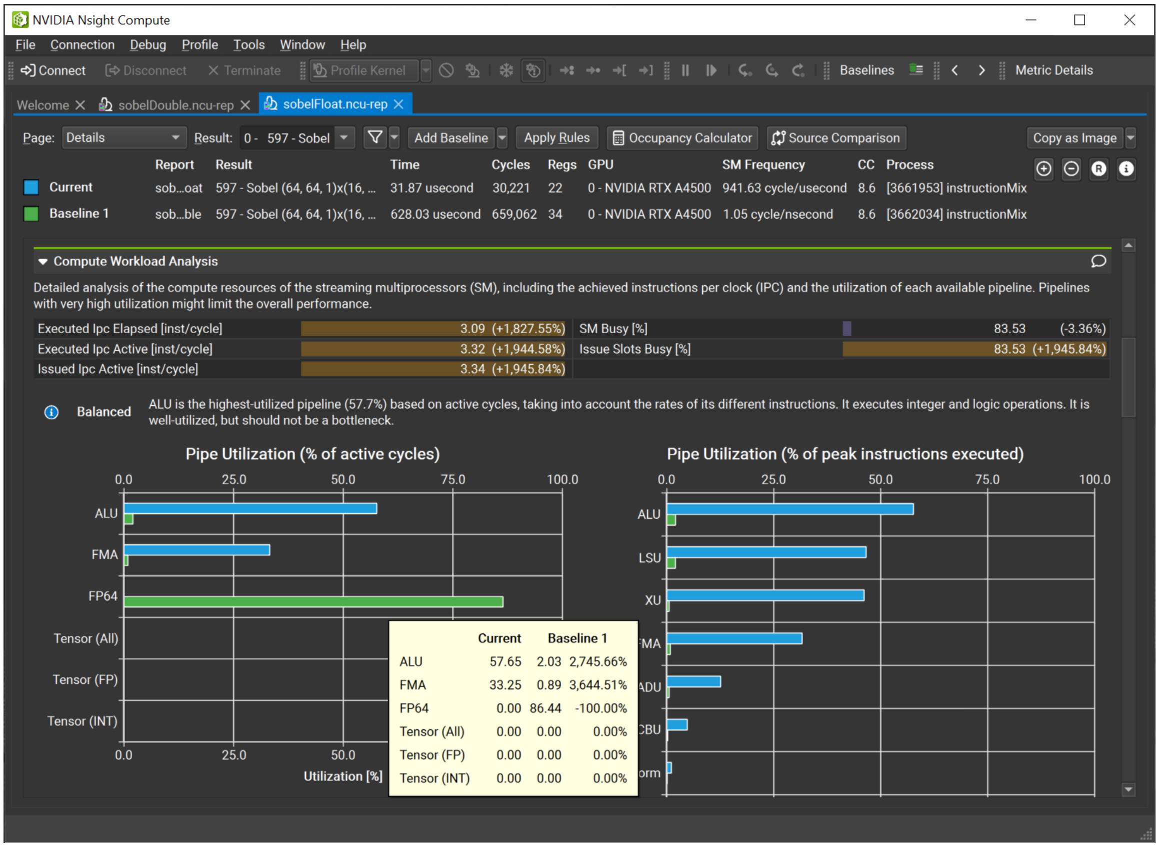
Compute (SM) Throughput [%]	77.67 (-8.89%)	Duration [usecond]	31.87 (-94.93%)
Memory Throughput [%]	46.36 (+2,249.69%)	Elapsed Cycles [cycle]	30,221 (-95.41%)
L1/TEX Cache Throughput [%]	49.87 (+2,392.35%)	SM Active Cycles [cycle]	27,911.98 (-95.70%)
L2 Cache Throughput [%]	21.20 (+1,835.25%)	SM Frequency [cycle/usecond]	941.63 (-10.26%)
DRAM Throughput [%]	22.19 (+1,900.00%)	DRAM Frequency [cycle/nsecond]	7.54 (-0.70%)

High Compute Throughput Compute is more heavily utilized than Memory: Look at the Compute Workload Analysis section to see what the compute pipelines are spending their time doing. Also, consider whether any computation is redundant and could be reduced or moved to look-up tables.

Roofline Analysis The ratio of peak float (fp32) to double (fp64) performance on this device is 64:1. The kernel achieved 11% of this device's fp32 peak performance and 0% of its fp64 peak performance. See the Kernel Profiling Guide for more details on roofline analysis.

	Current	Baseline 1	
Compute (SM) [%]	77.67	85.25	-8.89%
Memory [%]	46.36	1.97	2,249.69%

We can confirm from the **`Compute Workload Analysis`** section that no pipeline has a high utilization.


NVIDIA Nsight Compute
File Connection Debug Profile Tools Window Help
Connect Disconnect Terminate Profile Kernel Baselines Metric Details
Welcome sobelDouble.ncu-rep sobelFloat.ncu-rep
Page: Details Result: 0 - 597 - Sobel Add Baseline Apply Rules Occupancy Calculator Source Comparison Copy as Image
Pipe Utilization (% of active cycles)
0.0
25.0
50.0
75.0
100.0
ALU
FMA
FP64
Tensor (All)
Tensor (FP)
Tensor (INT)
Utilization [%]
Pipe Utilization (% of peak instructions executed)
ALU
LSU
XU
FMA
ADU
CBU
orm

	Report	Result	Time	Cycles	Regs	GPU	SM Frequency	CC	Process
🟦 Current	sob...oat	597 - Sobel (64, 64, 1)x(16, ...	31.87 usecond	30,221	22	0 - NVIDIA RTX A4500	941.63 cycle/usecond	8.6	[3661953] instructionMix
🟩 Baseline 1	sob...ble	597 - Sobel (64, 64, 1)x(16, ...	628.03 usecond	659,062	34	0 - NVIDIA RTX A4500	1.05 cycle/nsecond	8.6	[3662034] instructionMix

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]	3.09 (+1,827.55%)	SM Busy [%]	83.53 (-3.36%)
Executed Ipc Active [inst/cycle]	3.32 (+1,944.58%)	Issue Slots Busy [%]	83.53 (+1,945.84%)
Issued Ipc Active [inst/cycle]	3.34 (+1,945.84%)		

Balanced ALU is the highest-utilized pipeline (57.7%) based on active cycles, taking into account the rates of its different instructions. It executes integer and logic operations. It is well-utilized, but should not be a bottleneck.

	Current	Baseline 1	
ALU	57.65	2.03	2,745.66%
FMA	33.25	0.89	3,644.51%
FP64	0.00	86.44	-100.00%
Tensor (All)	0.00	0.00	0.00%
Tensor (FP)	0.00	0.00	0.00%
Tensor (INT)	0.00	0.00	0.00%

It shows a message that the pipe utilization is balanced and now the ALU is the highest-utilized pipeline (57.65%). From the pipeline utlization chart we see that the FP64 pipeline utlization is reduced from 86.44% to 0% and the single precision FMA pipeline utlization has increased from 0.89% to 33.25%.

Chapter 6.
RESOURCES

- Instruction Optimization section of the CUDA C++ Best Practices Guide
- Nsight Compute Documentation