



INSTRUCTION MIX SAMPLE

v2023.1.1 | December 2023



TABLE OF CONTENTS

Chapter 1. Introduction..... 1

Chapter 2. Application..... 2

Chapter 3. Configuration..... 3

Chapter 4. Initial version of the kernel.....4

Chapter 5. Updated version of the kernel.....9

Chapter 6. Resources..... 11

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 derivatives - 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.

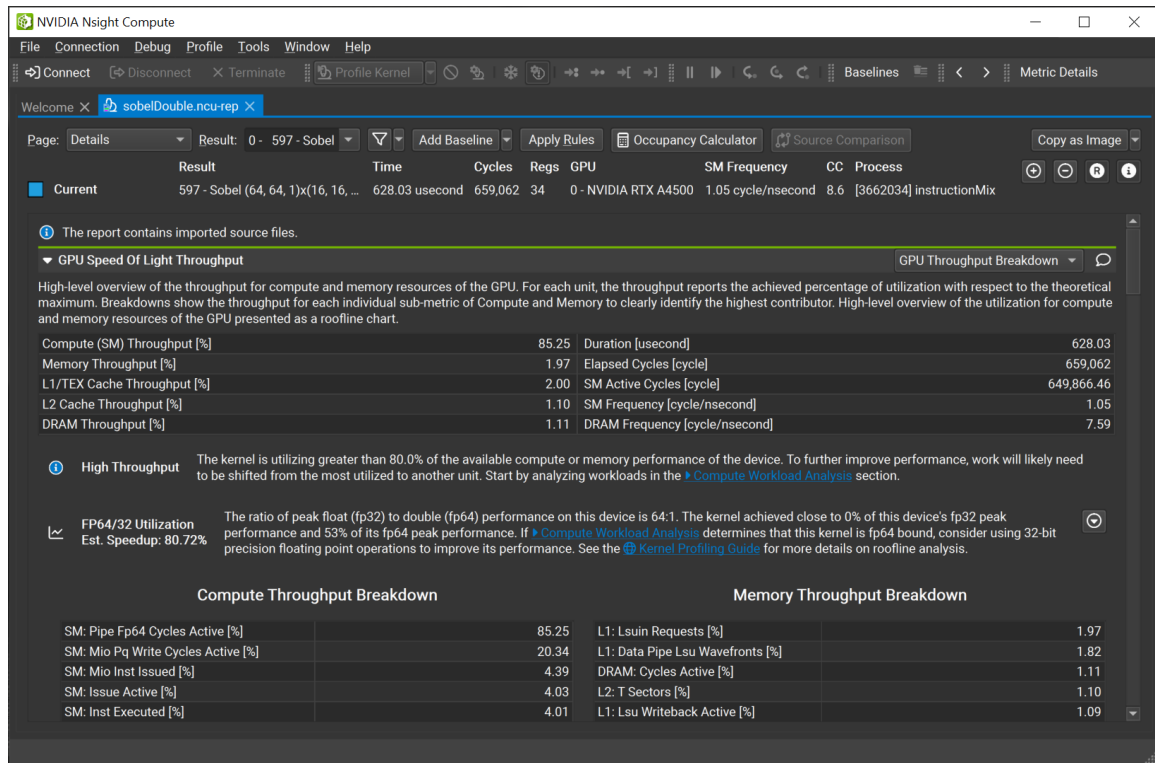
The screenshot shows the NVIDIA Nsight Compute interface. The top bar includes menus for File, Connection, Debug, Profile, Tools, Window, and Help. Below the menu is a toolbar with buttons for Connect, Disconnect, Terminate, Profile Kernel, and various performance metrics. The main content area displays the 'Summary' page for a kernel named 'Sobel'. It shows a table with columns for ID, Estimated Speedup, Function Name, Demangled Name, Duration, Runtime Improvement, Compute Throughput, and Memory Throughput. The table has one row with ID 0, Estimated Speedup 80.72, Function Name 'Sobel', Demangled Name 'void Sobel<double...>', Duration 628.03 usecond, Runtime Improvement 628.03, Compute Throughput 506.96, and Memory Throughput 85.25. Below the table, there are three performance optimization opportunities listed: 'FP64/32 Utilization' (Est. Speedup: 80.72%), 'FP64 Non-Fused Instructions' (Est. Speedup: 18.39%), and 'Uncoalesced Global Accesses' (Est. Speedup: 15.13%). Each opportunity includes a brief description and a link to the 'Details' page for more context.

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.

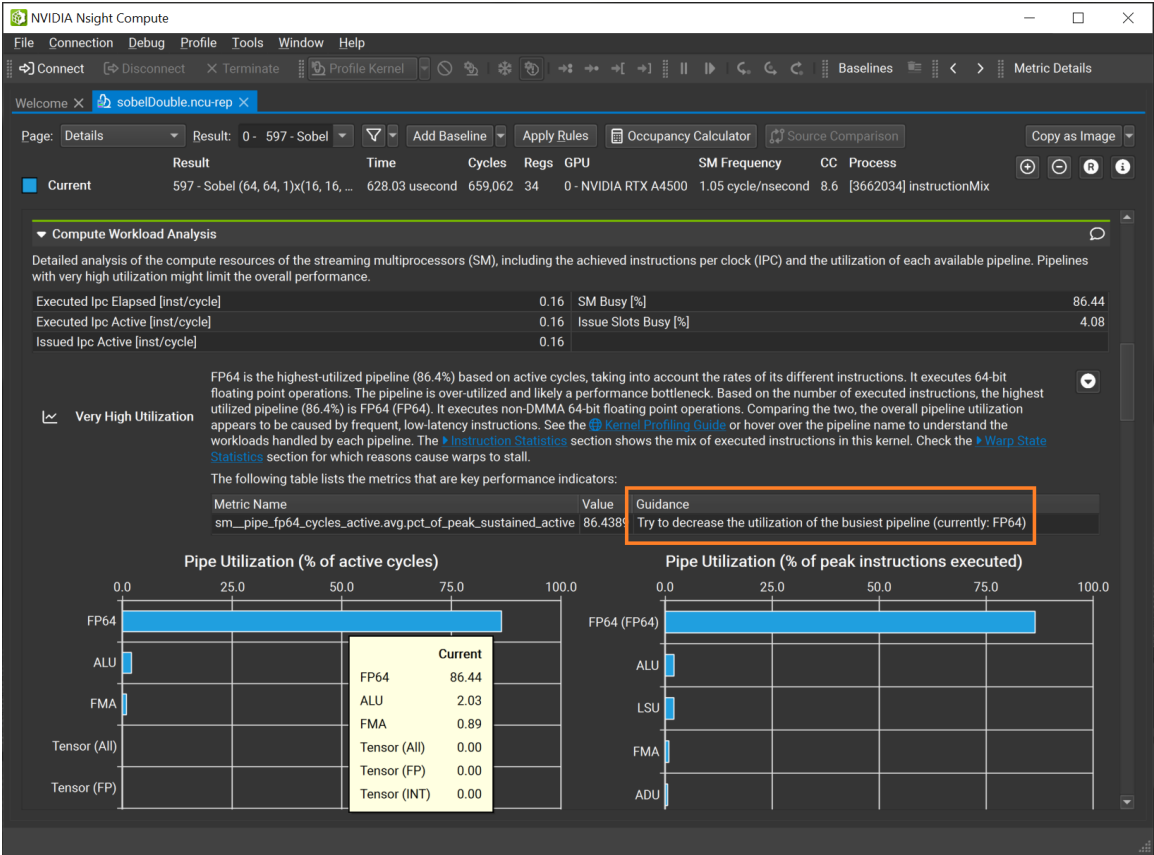
The initial version of the kernel has a duration of 628.03 microseconds and this is used as the baseline for further optimizations.



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 utilization of the FP64 pipeline.



Chapter 5.

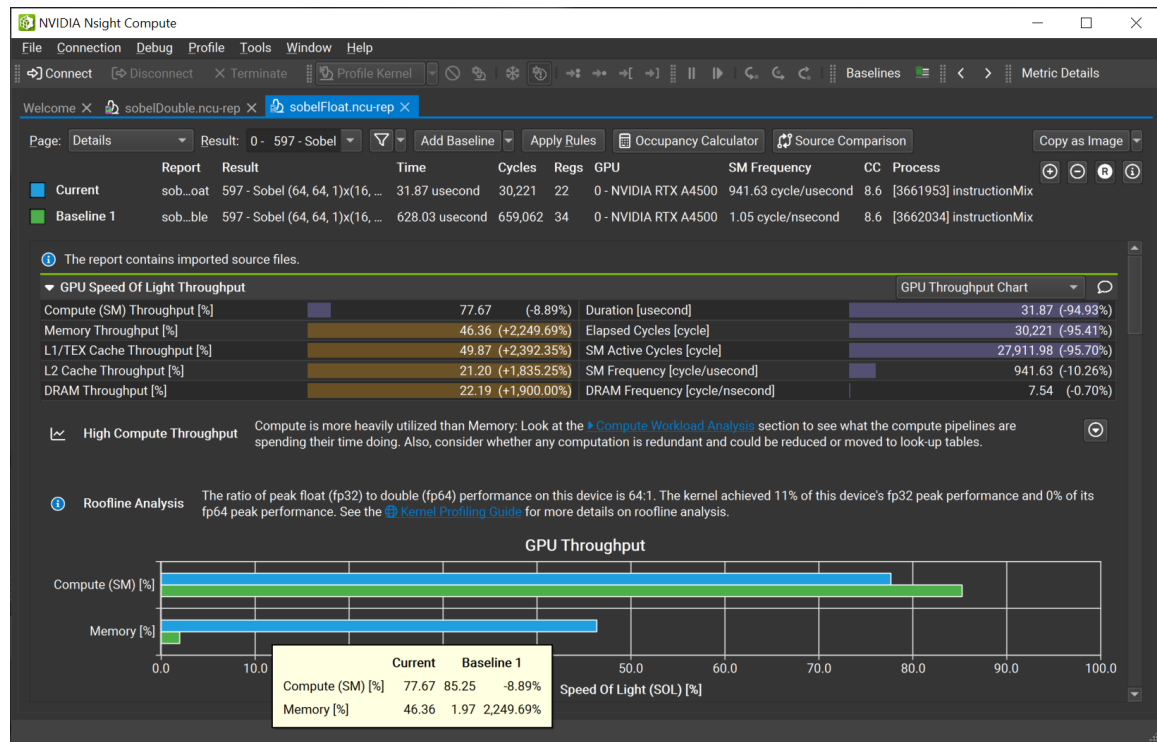
UPDATED VERSION OF THE KERNEL

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 receptive to minor differences in precision, switching the computations from double to single precision has no negative impact on its functionality.

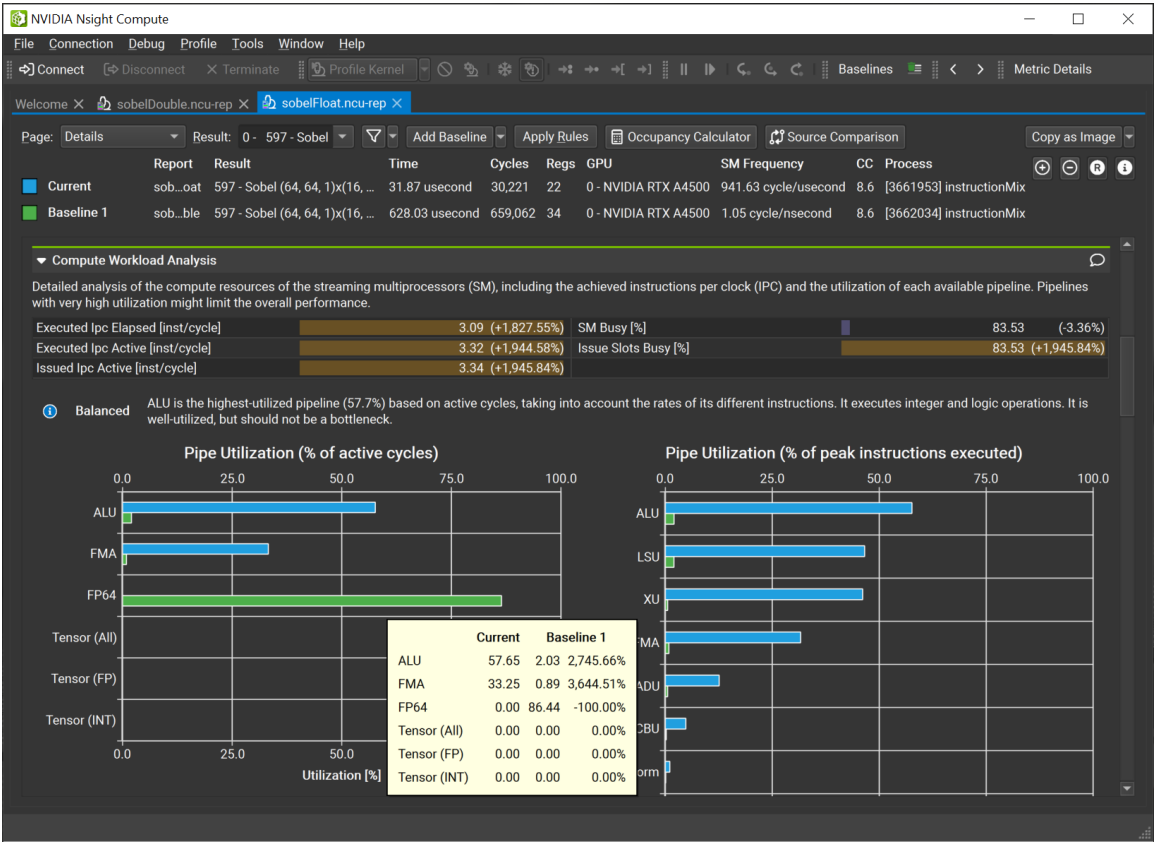
```
Sobel<float><<<grid, block>>>( pDstImage, pSrcImage, imgWidth,
imgHeight);
```

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.



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



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

Chapter 6.

RESOURCES

- ▶ [Instruction Optimization section of the CUDA C++ Best Practices Guide](#)
- ▶ [Nsight Compute Documentation](#)

Notice

ALL NVIDIA DESIGN SPECIFICATIONS, REFERENCE BOARDS, FILES, DRAWINGS, DIAGNOSTICS, LISTS, AND OTHER DOCUMENTS (TOGETHER AND SEPARATELY, "MATERIALS") ARE BEING PROVIDED "AS IS." NVIDIA MAKES NO WARRANTIES, EXPRESSED, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE MATERIALS, AND EXPRESSLY DISCLAIMS ALL IMPLIED WARRANTIES OF NONINFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE.

Information furnished is believed to be accurate and reliable. However, NVIDIA Corporation assumes no responsibility for the consequences of use of such information or for any infringement of patents or other rights of third parties that may result from its use. No license is granted by implication of otherwise under any patent rights of NVIDIA Corporation. Specifications mentioned in this publication are subject to change without notice. This publication supersedes and replaces all other information previously supplied. NVIDIA Corporation products are not authorized as critical components in life support devices or systems without express written approval of NVIDIA Corporation.

Trademarks

NVIDIA and the NVIDIA logo are trademarks or registered trademarks of NVIDIA Corporation in the U.S. and other countries. Other company and product names may be trademarks of the respective companies with which they are associated.

Copyright

© 2023-2023 NVIDIA Corporation and affiliates. All rights reserved.

This product includes software developed by the Syncro Soft SRL (<http://www.sync.ro/>).