UNCOALESCED GLOBAL ACCESSES SAMPLE

v2022.5.0 | February 2023

TABLE OF CONTENTS

Chapter 1. INTRODUCTION

This sample profiles a memory-bound CUDA kernel which does a simple computation on an array of double3 data type in global memory using the Nsight Compute profiler. The profiler is used to analyze and identify the memory accesses which are uncoalesced and result in inefficient DRAM accesses.

Global memory accesses on a GPU

Global memory resides in device memory and device memory is accessed via 32, 64, or 128-byte memory transactions.

When a warp executes an instruction that accesses global memory, it coalesces the memory accesses of the threads within the warp into one or more of these memory transactions depending on the size of the data accessed by each thread and the distribution of the memory addresses across the threads. If global memory accesses of the threads within a warp cannot be combined into the same memory transaction then we refer to these as uncoalesced global memory accesses. In general, the more transactions are necessary, the more unused bytes are transferred in addition to the bytes accessed by the threads, reducing the instruction throughput accordingly. For example, if a 32-byte memory transaction is generated for each thread's 4-byte access, throughput is divided by 8.

Chapter 2. APPLICATION

The sample CUDA application adds a floating point constant to an input array of 1,048,576 (1024*1024) double3 elements in global memory and generates an output array of double3 in global memory of the same size. double3 is a 24-byte built-in vector type which is a structure containing 3 double precision floating point values:

```
struct
{
    double x, y, z;
};
```

The uncoalescedGobalAccesses sample is available with Nsight Compute under <nsight-compute-install-directory>/extras/samples/uncoalescedGlobalAccesses.

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 A2000 (Ampere GA106) GPU
- Nsight Compute version: 2022.1.1

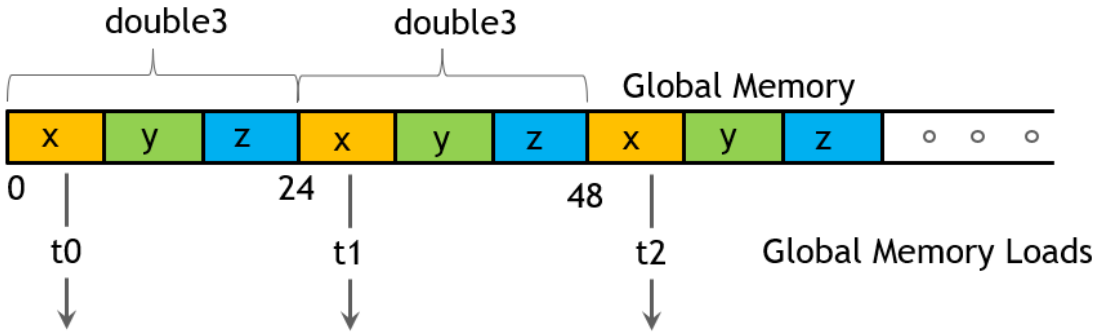
The Nsight Compute UI screen shots in the document are taken by opening the profiling reports on a Windows 10 system.

Chapter 4.
INITIAL VERSION OF THE KERNEL

The initial version of the sample code provides a naive implementation for the kernel which adds a floating point constant to an input array of double3.

```
__global__ void addConstDouble3(int numElements, double3 *d_in, double k,
 double3 *d_out)
{
    int index = blockIdx.x * blockDim.x + threadIdx.x;
    if (index < numElements)
    {
        double3 a = d_in[index];
        a.x += k;
        a.y += k;
        a.z += k;
        d_out[index] = a;
    }
}
```

The instruction **`a = d_in[index]`** in the kernel code results in each thread in a warp accessing global memory 24-bytes apart. In the first step all threads request a load for **`d_in[index].x`** as shown in the following diagram. In the second step a load for **`d_in[index].y`** and in the third step a load for **`d_in[index].z`** is made by all threads.



The instruction **`d_out[index] = a;`** has a similar multistep storage pattern.

Profile the initial version of the kernel

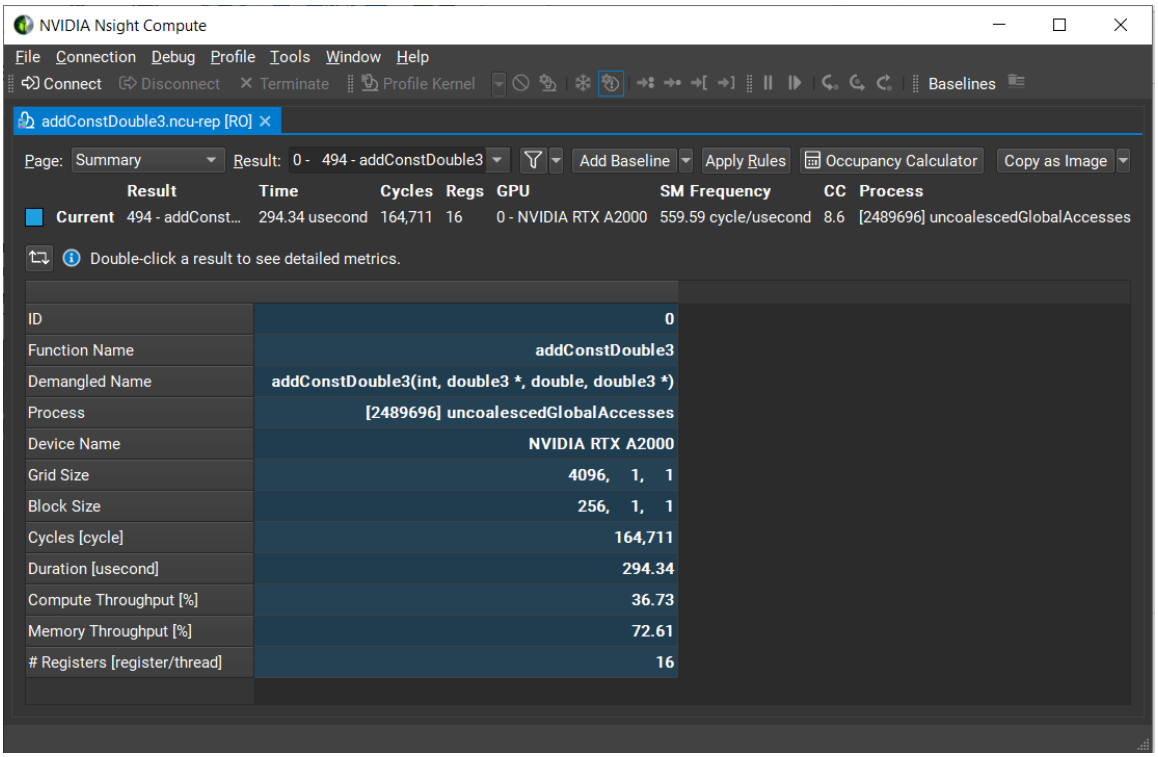
There are multiple ways to profile kernels with Nsight Compute. For full details see the Nsight Compute Documentation. One example workflow to follow for this sample:

- 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

Summary page

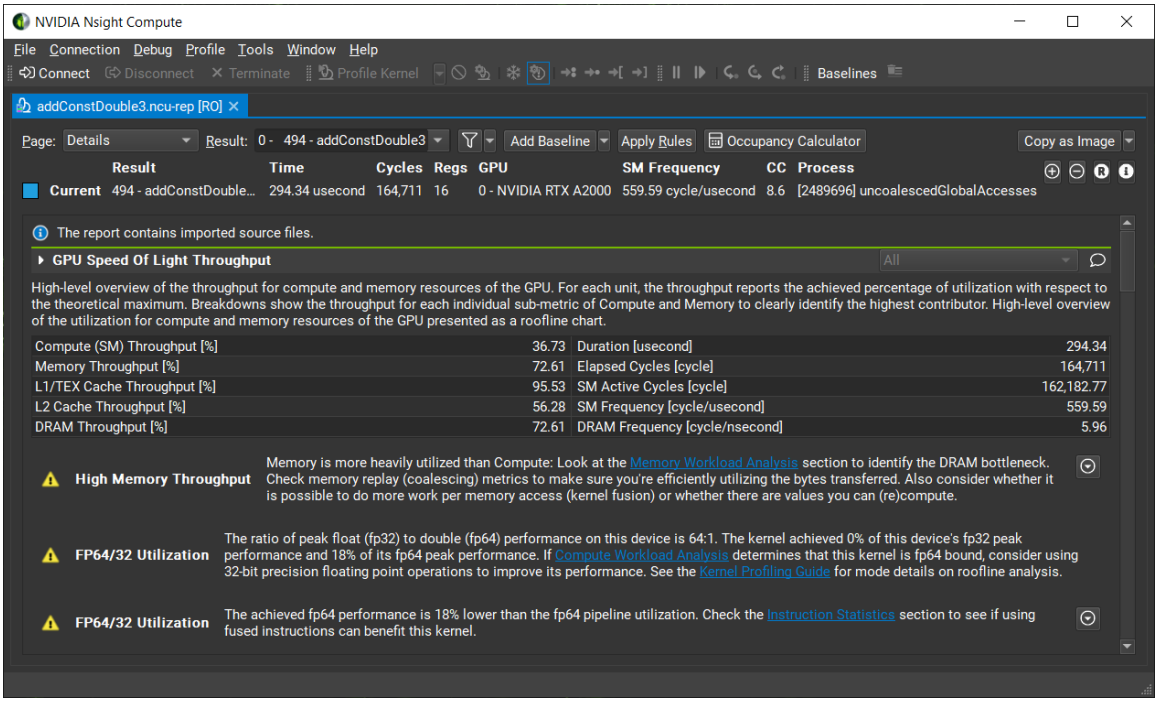
All kernels in the application are profiled and the summary page is displayed. The kernel launch parameters, cycles, duration, compute and memory throughput for each kernel are shown. In this sample we have only one kernel launch.

The duration for this initial version of the kernel is 294 micro seconds and this is used as the baseline for further optimizations.



Details page - GPU Speed Of Light Throughput

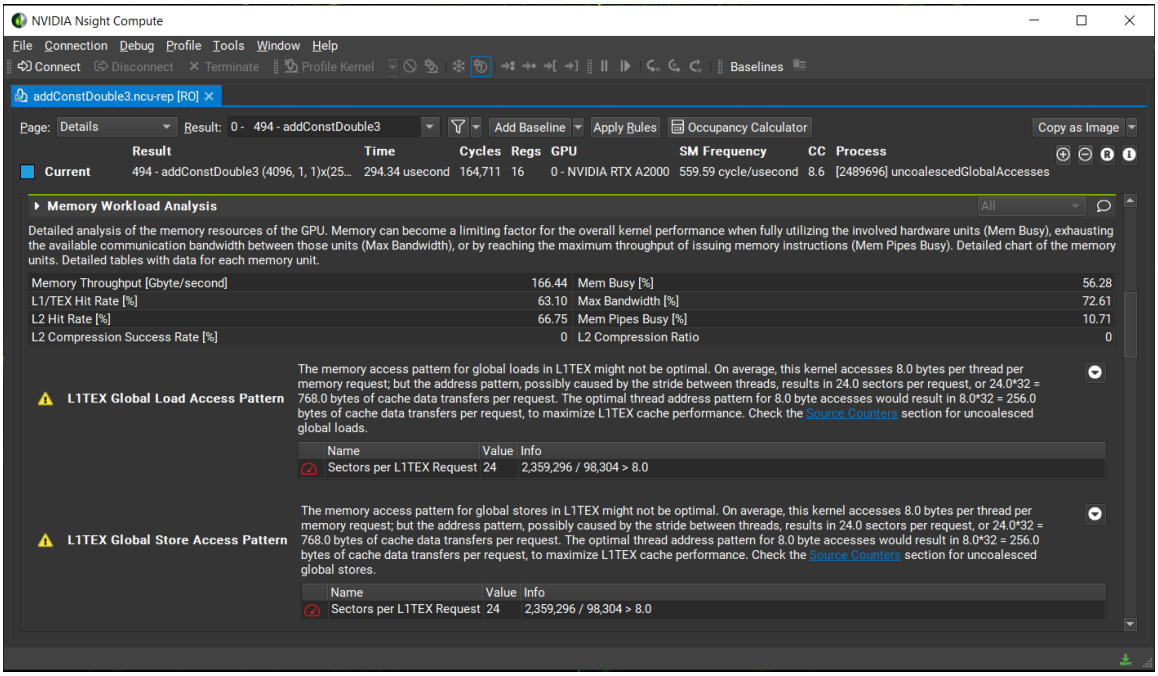
The details page "GPU Speed Of Light Throughput" section provides a high-level overview of the throughput for compute and memory resources of the GPU used by the kernel.



For this kernel it shows a hint for High Memory Throughput and suggests looking at the memory workload analysis section. Click on Memory Workload Analysis.

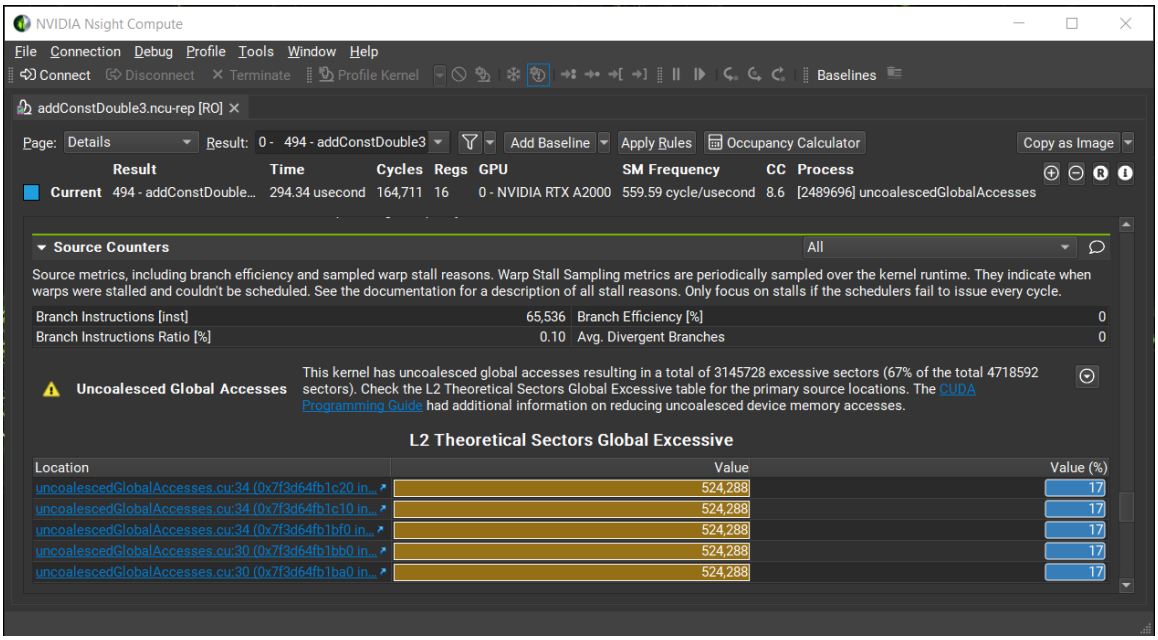
Details page - Memory Workload Analysis section

The Memory Workload Analysis shows hints for L1TEX Global store and load access patterns. The description and focus metrics of these performance issues describe how more sectors than necessary are being accessed from memory. A sector is an aligned 32-byte chunk of memory in a cache line or device memory. These additional sector accesses are caused by uncoalesced memory accesses and can negatively impact performance. In this case, for the load or store instructions, each thread is accessing a double (8 bytes) and there are 32 threads in a warp. Therefore, each memory request from a warp should ideally access 256 bytes (8 x 32), which is 8 sectors. However, in this unoptimized version, we see 24 sectors per request. It suggests checking the Source Counters section for uncoalesced global stores and loads. Click on the Source Counters link.



Details page - Source Counters section

The Source Counters section shows a hint for "Uncoalesced Global Accesses". It explains that the metric “L2 Theoretical Sectors Global Excessive” is the indicator for uncoalesced accesses. The table for this metric lists the source lines with the highest value. Click on one of the source lines to view the kernel source at which the bottleneck occurs.



Source page

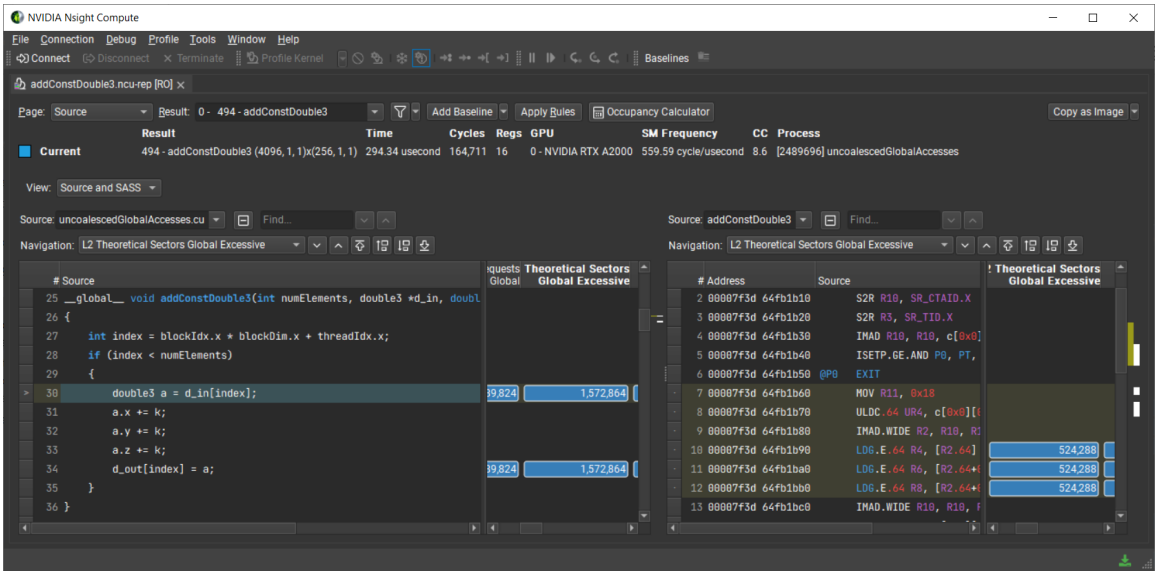
The CUDA source and SASS(GPU Assembly) for the kernel is shown side by side. When opening the Source page from Source Counters section, the Navigation metric is automatically filled in to match, in this case "L2 Theoretical Sectors Global Excessive". You can see this by the bolding in the column header. The source line at which the bottleneck occurs is highlighted.

It shows uncoalesced global memory load accesses at line #30:

```
        double3 a = d_in[index];
```

It shows uncoalesced global memory store accesses at line #34:

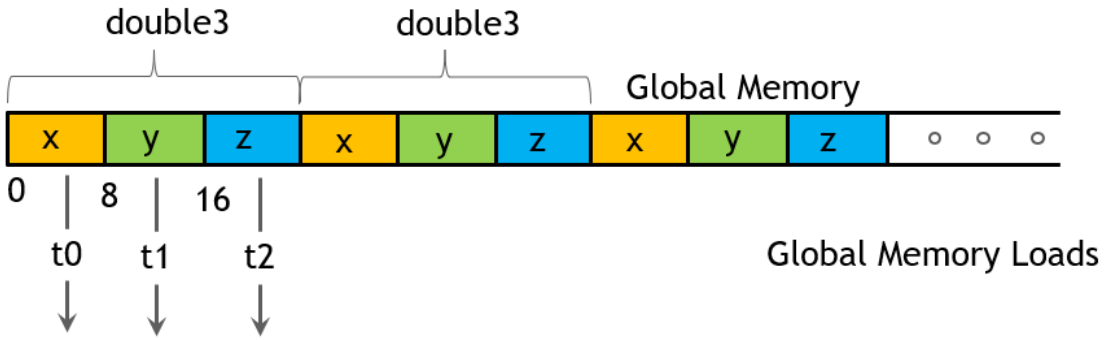
```
        d_out[index] = a;
```



Chapter 5.
UPDATED VERSION OF THE KERNEL

Considering the uncoalesced accesses reported by the profiler we analyze the global load access pattern. Each thread executes 3 reads for the three double values in double3.

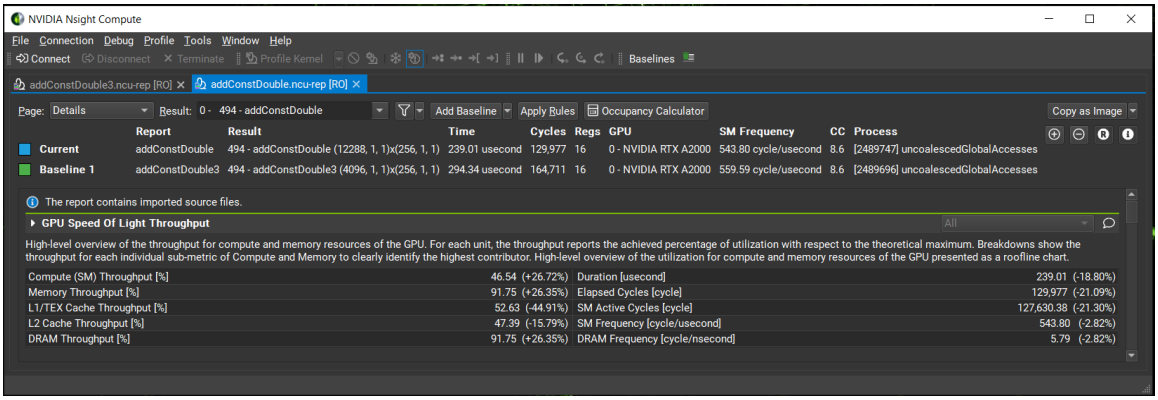
We can treat the double3 array as a double array and each thread can process one double instead of one double3. With this change threads in a warp access consecutive double values and both loads and stores are coalesced.



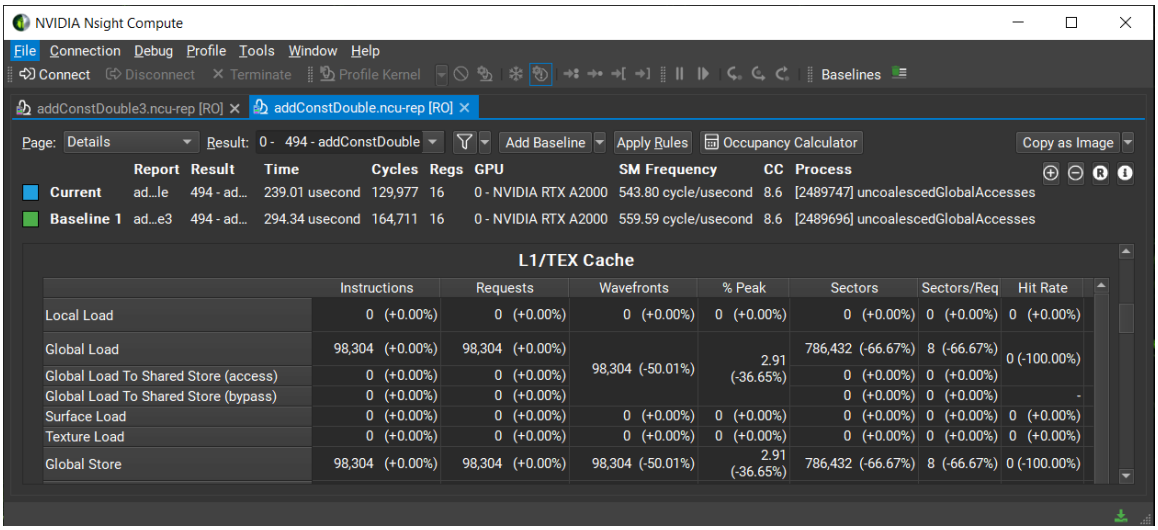

```
__global__ void addConstDouble(int numElements, double *d_in, double k, double
 *d_out)
{
    int index = blockIdx.x * blockDim.x + threadIdx.x;
    if (index < numElements)
    {
        d_out[index] = d_in[index] + k;
    }
}
```

Profile the updated kernel

The kernel duration has reduced from 294 microseconds to 239 microseconds. We can set a baseline to the initial version of the kernel and compare the profiling results.



We can confirm that the global memory accesses are coalesced. In the L1/TEX Cache metrics table under the Memory workload analysis section we see that the "Sectors/Req" metric value is 8 for both global loads and global stores.



Chapter 6. RESOURCES

- GPU Technology Conference 2021 talk S32089: Requests, Wavefronts, Sectors Metrics: Understanding and Optimizing Memory-Bound Kernels with Nsight Compute
- Nsight Compute Documentation

Notice

Trademarks

Copyright