



# UNCOALESCED GLOBAL ACCESSES SAMPLE

v2022.5.0 | July 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..... 10
- Chapter 6. Resources..... 12

# 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 uncoalescedGlobalAccesses 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: 2022.4.0

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

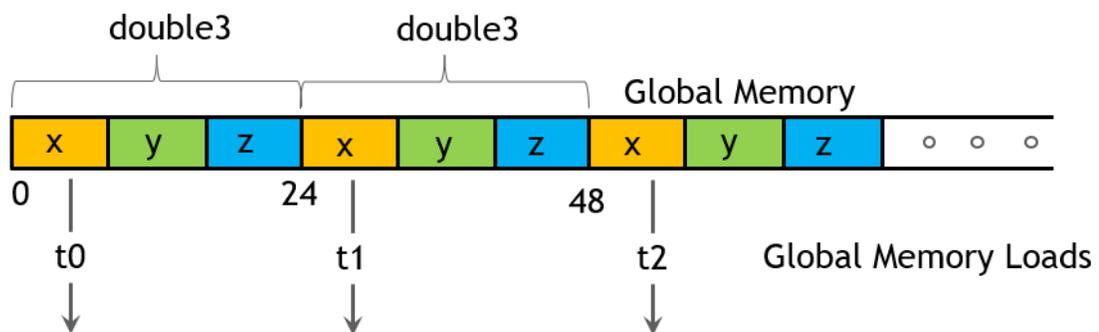
# Chapter 4.

## 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 292.99 micro seconds and this is used as the baseline for further optimizations.

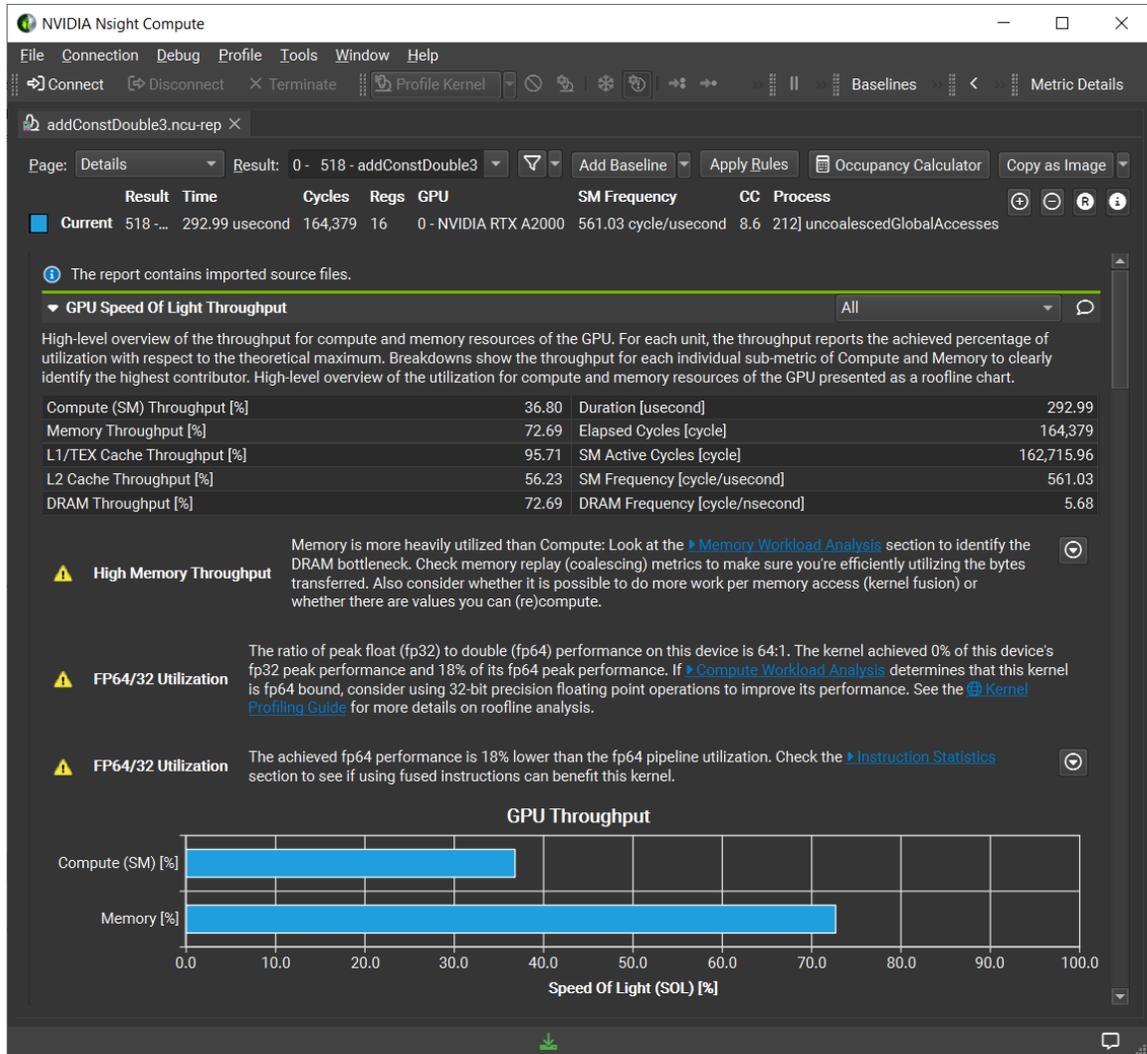
Result	Time	Cycles	Regs	GPU	SM Frequency	CC Process
Current 518 - addConstDouble3	292.99 usecond	164,379	16	0 - NVIDIA RTX A2000	561.03 cycle/usecond	8.6 [2212] uncoalescedGlobalAccesses

Double-click a result to see detailed metrics.

ID	0
Issues Detected	11
Function Name	addConstDouble3
Demangled Name	addConstDouble3(int, double3 *, double, double3 *)
Process	[2212] uncoalescedGlobalAccesses
Device Name	NVIDIA RTX A2000
Grid Size	4096, 1, 1
Block Size	256, 1, 1
Cycles [cycle]	164,379
Duration [usecond]	292.99
Compute Throughput [%]	36.80
Memory Throughput [%]	72.69
# Registers [register/thread]	16

## 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.

The screenshot displays the NVIDIA Nsight Compute interface for a kernel named 'addConstDouble3.ncu-rep'. The main performance metrics are as follows:

Result	Time	Cycles	Regs	GPU	SM Frequency	CC	Process
Current	518 ...	292.99 usecond	164,379	16	0 - NVIDIA RTX A2000	561.03 cycle/usecond	8.6 212] uncoalescedGlobalAccesses

Other metrics include: Issued lpc Active [inst/cycle] = 0.16. The balanced status indicates that FP64 is the highest-utilized pipeline (37.2%) based on active cycles, taking into account the rates of its different instructions. It executes 64-bit floating point operations. It is well-utilized, but should not be a bottleneck.

**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). Detailed chart of the memory units. Detailed tables with data for each memory unit.

Memory Throughput [Gbyte/second]	167.02	Mem Busy [%]	56.23
L1/TEX Hit Rate [%]	62.97	Max Bandwidth [%]	72.69
L2 Hit Rate [%]	66.61	Mem Pipes Busy [%]	10.73
L2 Compression Success Rate [%]	0	L2 Compression Ratio	0

**L1TEX Global Load Access Pattern**

The memory access pattern for global loads in L1TEX might not be optimal. On average, this kernel accesses 8.0 bytes per thread per memory request; but the address pattern, possibly caused by the stride between threads, results in 24.0 sectors per request, or  $24.0 \times 32 = 768.0$  bytes of cache data transfers per request. The optimal thread address pattern for 8.0 byte accesses would result in  $8.0 \times 32 = 256.0$  bytes of cache data transfers per request, to maximize L1TEX cache performance. Check the [Source Counters](#) section for uncoalesced global loads.

Name	Value	Info
Sectors per L1TEX Request	24	2,359,296 / 98,304 > 8.0

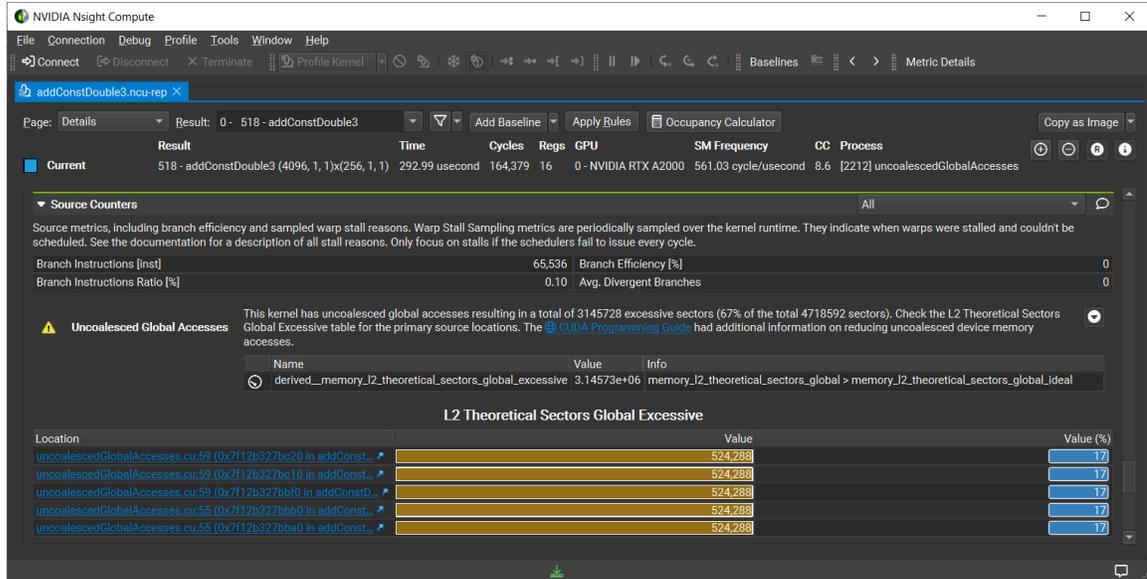
**L1TEX Global Store Access Pattern**

The memory access pattern for global stores in L1TEX might not be optimal. On average, this kernel accesses 8.0 bytes per thread per memory request; but the address pattern, possibly caused by the stride between threads, results in 24.0 sectors per request, or  $24.0 \times 32 = 768.0$  bytes of cache data transfers per request. The optimal thread address pattern for 8.0 byte accesses would result in  $8.0 \times 32 = 256.0$  bytes of cache data transfers per request, to maximize L1TEX cache performance. Check the [Source Counters](#) section for uncoalesced global stores.

Name	Value	Info
Sectors per L1TEX Request	24	2,359,296 / 98,304 > 8.0

## 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 #55:

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

It shows uncoalesced global memory store accesses at line #59:

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

The screenshot displays the NVIDIA Nsight Compute interface for a kernel named 'addConstDouble3.ncu-rep'. The top status bar shows the following performance metrics:

Result	Time	Cycles	Regs	GPU	SM Frequency	CC	Process
518 - addConstDouble3 (4096, 1, 1)x(256, 1, 1)	292.99 usecond	164,379	16	0 - NVIDIA RTX A2000	561.03 cycle/usecond	8.6	[212] uncoalescedGlobalAccesses

The main window is split into three panes:

- Source Code:** Shows the C++ kernel code for `addConstDouble3`. Lines 55 and 59 are highlighted, showing memory accesses: `double3 a = d_in[index];` and `d_out[index] = a;`.
- Accesses:** A table showing memory access details for the highlighted lines.
 

#	Source	Address Space	Access Operation	Access Size	L2 Theoretical Sectors Global Excessive
55	<code>double3 a = d_in[index];</code>	Global(3)	Load(3)	64(3)	1,572,864
59	<code>d_out[index] = a;</code>	Global(3)	Store(3)	64(3)	1,572,864
- Disassembly:** A table showing the assembly instructions for the kernel.
 

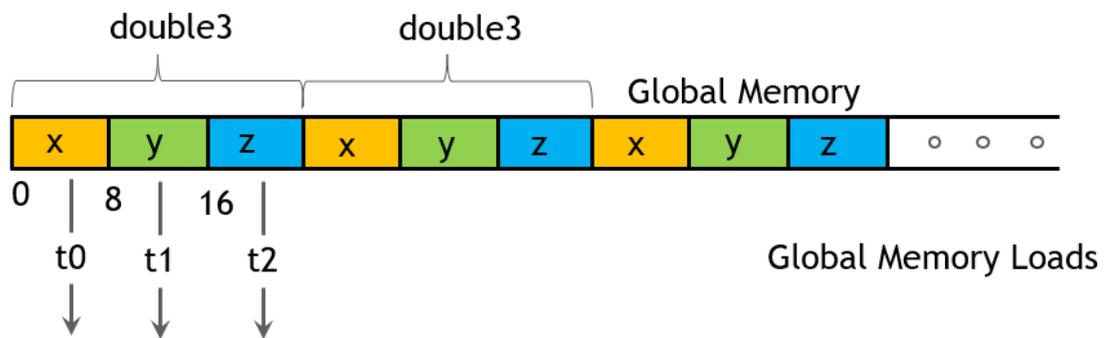
#	Address	Source	Access Size	L2 Theoretical Sectors Global Excessive
1	00007f12 b327bb00	<code>MOV R1, c[0x0]</code>		
2	00007f12 b327bb10	<code>S2R R10, SR_CTA</code>		
3	00007f12 b327bb20	<code>S2R R3, SR_TID</code>		
4	00007f12 b327bb30	<code>IMAD R10, R10, c</code>		
5	00007f12 b327bb40	<code>ISETP_GE_AND P0,</code>		
6	00007f12 b327bb50	<code>EXIT</code>		
7	00007f12 b327bb60	<code>MOV R11, 0x18</code>		
8	00007f12 b327bb70	<code>ULDC_64 UR4, c[</code>		
9	00007f12 b327bb80	<code>IMAD_WIDE R2, R</code>		
10	00007f12 b327bb90	<code>LDG_E_64 R4, [R</code>	64	524,288
11	00007f12 b327bba0	<code>LDG_E_64 R6, [R</code>	64	524,288
12	00007f12 b327bbb0	<code>LDG_E_64 R8, [R</code>	64	524,288
13	00007f12 b327bbc0	<code>IMAD_WIDE R10, f</code>		
14	00007f12 b327bbd0	<code>DADD R4, R4, c[</code>		
15	00007f12 b327bbe0	<code>DADD R6, R6, c[</code>		
16	00007f12 b327bbf0	<code>STG_E_64 [R10, 6</code>	64	524,288

# 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.

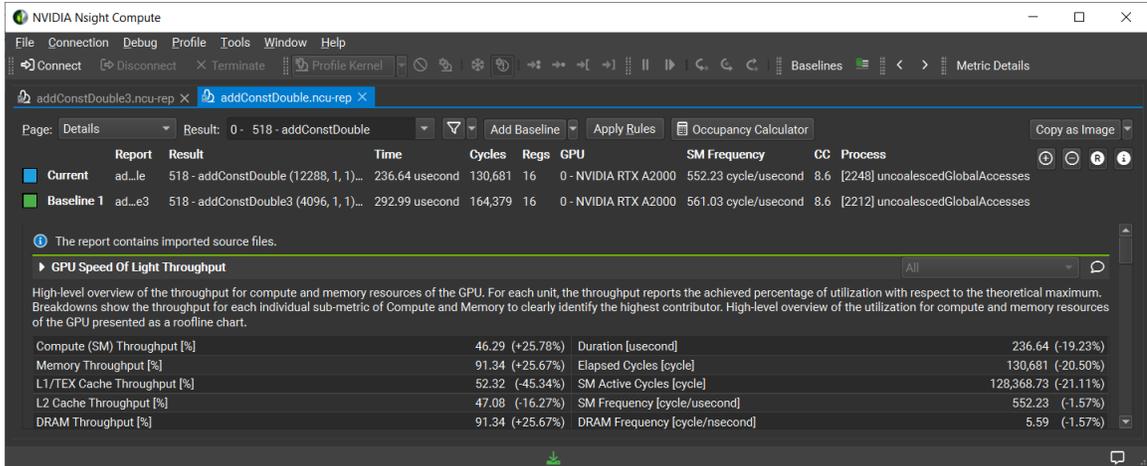
We can treat the double3 array as a double array and each thread can process one double instead of one double3. With this change threads in a warp access consecutive double values and both loads and stores are coalesced.



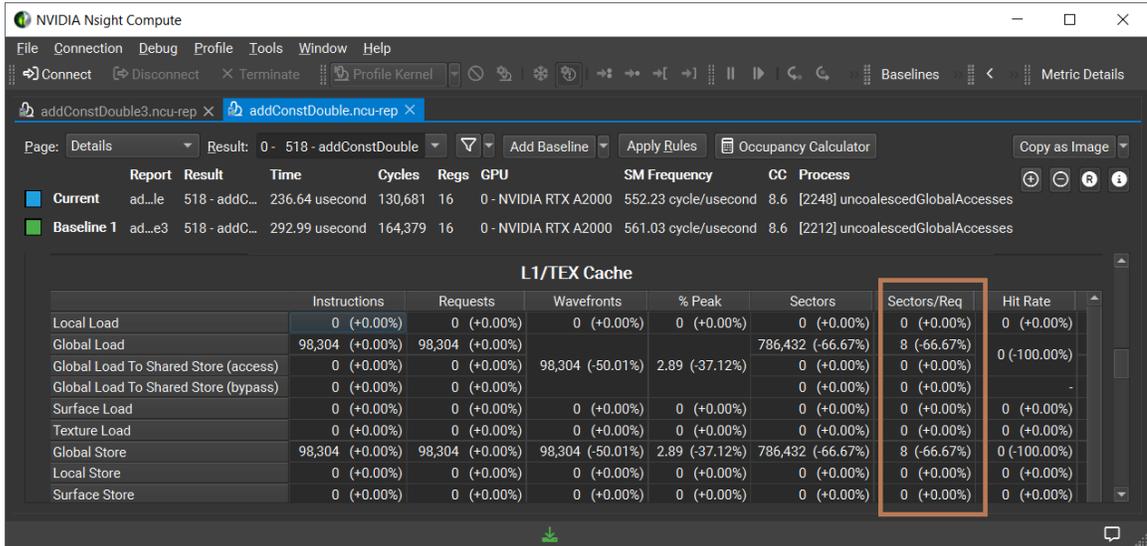
```
__global__ void addConstDouble(int numElements, double *d_in, double k, double *d_out)
{
    int index = blockIdx.x * blockDim.x + threadIdx.x;
    if (index < numElements)
    {
        d_out[index] = d_in[index] + k;
    }
}
```

### Profile the updated kernel

The kernel duration has reduced from 293.99 microseconds to 236.64 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**

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**

© 2022-2023 NVIDIA Corporation and affiliates. All rights reserved.

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