



# SHARED MEMORY BANK CONFLICTS SAMPLE

v2023.1.0 | September 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..... 13

# Chapter 1.

## INTRODUCTION

This sample profiles a CUDA kernel which transposes an  $N \times N$  square matrix of float elements in global memory using the Nsight Compute profiler. To avoid uncoalesced global memory accesses this kernel reads the data into shared memory. The profiler is used to analyze and identify the shared memory bank conflicts which result in inefficient shared memory accesses.

### Shared memory accesses on a GPU

Shared memory is located on-chip, so it has much higher bandwidth and much lower latency than either local or global memory. Shared memory can be shared across a compute Cooperative Thread Array (CTA). In CUDA, CTAs are referred to as Thread Blocks. Compute CTAs attempting to share data across threads via shared memory must use synchronization operations (such as `__syncthreads()`) between stores and loads to ensure data written by any one thread is visible to other threads in the CTA.

Shared memory has 32 banks that are organized such that successive 32-bit words map to successive banks that can be accessed simultaneously. Any 32-bit memory read or write request made of 32 addresses that fall in 32 distinct memory banks can therefore be serviced simultaneously, yielding an overall bandwidth that is 32 times as high as the bandwidth of a single request. However, if two addresses of a memory request fall in the same memory bank, there is a bank conflict and the access has to be serialized. The exception to this rule is when all threads read the same shared memory address, which results in a broadcast where the data at that address is sent to all threads in one transaction.

To get maximum performance, it is therefore important to understand how memory addresses map to memory banks in order to schedule the memory requests so as to minimize bank conflicts.

## Chapter 2. APPLICATION

The sample CUDA application transposes a matrix of floats. The input and output matrices are at separate memory locations. For simplicity it only handles square matrices whose dimensions are integral multiples of 32, the tile size.

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

# 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: 2023.3.0

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 kernel `transposeCoalesced` uses shared memory to ensure that global memory accesses for loading data from the input matrix `idata` and storing data in the output matrix `odata` are coalesced. The matrix is sub-divided into tiles of size 32 x 32. The tile size is defined as:

```
#define TILE_DIM    32
```

For simplicity the code only handles square matrices whose dimensions are integral multiples of 32, the tile size. Each block transposes a tile of 32 x 32 elements. Each thread in the block transposes `TILE_DIM/BLOCK_ROWS` i.e. 4 elements, where `BLOCK_ROWS` is defined as:

```
#define BLOCK_ROWS  8
```

`TILE_DIM` must be an integral multiple of `BLOCK_ROWS`.

The way to avoid uncoalesced global memory access is to read the data into shared memory, and have each warp access noncontiguous locations in shared memory in order to write contiguous data to `odata`. The above procedure requires that each element in a tile be accessed by different threads, so a `__syncthreads ()` call is required to ensure

that all reads from **idata** to shared memory have completed before writes from shared memory to **odata** commence.

```

__global__ void transposeCoalesced(float* odata, float* idata, int width, int
height)
{
    __shared__ float tile[TILE_DIM][TILE_DIM];

    int xIndex = blockIdx.x * TILE_DIM + threadIdx.x;
    int yIndex = blockIdx.y * TILE_DIM + threadIdx.y;
    int indexIn = xIndex + yIndex*width;

    xIndex = blockIdx.y * TILE_DIM + threadIdx.x;
    yIndex = blockIdx.x * TILE_DIM + threadIdx.y;
    int indexOut = xIndex + yIndex*height;

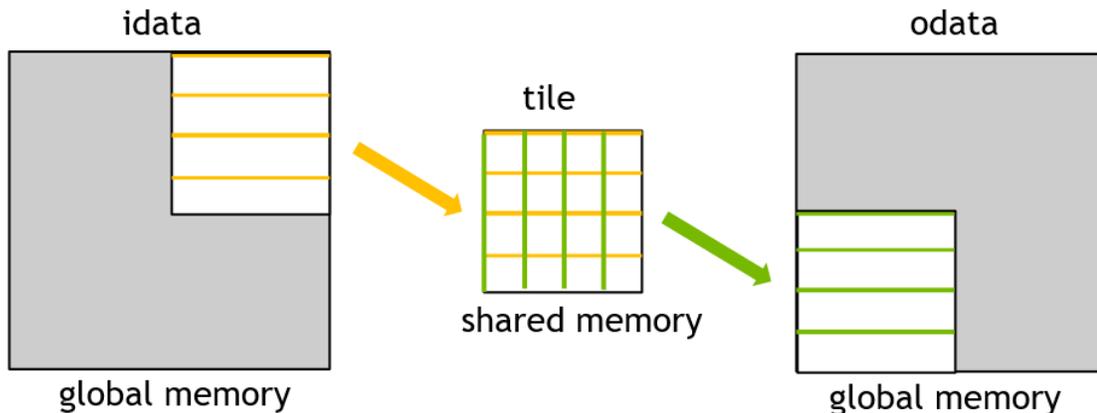
    for (int i = 0; i < TILE_DIM; i += BLOCK_ROWS)
    {
        tile[threadIdx.y + i][threadIdx.x] = idata[indexIn + i * width];
    }

    __syncthreads();

    for (int i = 0; i < TILE_DIM; i += BLOCK_ROWS)
    {
        odata[indexOut + i * height] = tile[threadIdx.x][threadIdx.y + i];
    }
}

```

A depiction of the data flow of a warp in the coalesced transpose kernel is given below. The warp writes four rows of the **idata** matrix tile to the shared memory 32x32 array "tile" indicated by the yellow line segments. After a **\_\_syncthreads()** call to ensure all writes to tile are completed, the warp writes four columns of tile to four rows of an **odata** matrix tile, indicated by the green line segments.



### 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
- ▶ 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. In this sample we have only one kernel launch.

The duration for this initial version of the kernel is 5.59 milliseconds and this is used as the baseline for further optimizations.

The screenshot displays the NVIDIA Nsight Compute interface. At the top, there's a menu bar with options like File, Connection, Debug, Profile, Tools, Window, and Help. Below that, a toolbar contains various icons for actions like Connect, Disconnect, Terminate, Profile Kernel, and Baselines. The main content area shows a summary for a kernel named 'transposeCoalesced.ncu-rep'. A table lists the results, with the current result (ID 0) showing an estimated speedup of 93.80%, a duration of 5.59 ms, and a compute throughput of 23.09. Below the table, there are three performance optimization opportunities, each with a brief description and an estimated speedup percentage.

ID	Estimated Speedup	Function Name	Demangled Name	Duration	Runtime Improvement	Compute Throughput
0	93.80	transposeCoalesced	transposeCoalesced(float *, float *, int, int)	5.59	5.24	23.09

The following performance optimization opportunities were discovered for this result. Follow the rule links to see more context on the Details page.  
Note: Speedup estimations are experimental and might overestimate optimization potential.

- Uncoalesced Shared Accesses**  
Est. Speedup: 93.80%  
This kernel has uncoalesced shared accesses resulting in a total of 65011712 excessive wavefronts (94% of the total 69206016 wavefronts). Check the L1 Wavefronts Shared Excessive table for the primary source locations. The [CUDA Best Practices Guide](#) has an example on optimizing shared memory accesses.
- Shared Load Bank Conflicts**  
Est. Speedup: 89.06%  
The memory access pattern for shared loads might not be optimal and causes on average a 32.2-way bank conflict across all 2097152 shared load requests. This results in 65011712 bank conflicts, which represent 96.37% of the overall 67458953 wavefronts for shared loads. Check the [Source Counters](#) section for uncoalesced shared loads.
- Mio Throttle Stalls**  
Est. Speedup: 54.67%  
On average, each warp of this kernel spends 85.4 cycles being stalled waiting for the MIO (memory input/output) instruction queue to be not full. This stall reason is high in cases of extreme utilization of the MIO pipelines, which include special math instructions, dynamic branches, as well as shared memory instructions. When caused by shared memory accesses, trying to use fewer but wider loads can reduce pipeline pressure. This stall type represents about 54.7% of the total average of 156.2 cycles between issuing two instructions.

For this kernel it shows three performance opportunities. The topmost performance opportunity is for **Uncoalesced Shared Accesses** and it suggests checking the **L1 Wavefronts Shared Excessive** table for the primary source locations. Click on **Uncoalesced Shared Accesses** rule link to see more context on the **Details** page. It opens the **Source Counters** section on the **Details** page.

## Details page

The Source Counters section table for the metric **L1 Wavefronts Shared Excessive** which is an indicator for shared memory bank conflicts lists the source lines with the highest value.

The screenshot shows the NVIDIA Nsight Compute interface. The top bar indicates the kernel name 'transposeCoalesced.ncu-rep'. The main display area shows the 'Source Counters' section for the metric 'L1 Wavefronts Shared Excessive'. The 'Uncoalesced Shared Accesses' section shows an estimated speedup of 93.80%. The 'L1 Wavefronts Shared Excessive' table lists the following data:

Location	Value	Value (%)
<a href="#">sharedBankConflicts.cu:95 (0x7fc37325d430 in _Z1...</a>	16,252,928	25
<a href="#">sharedBankConflicts.cu:95 (0x7fc37325d420 in _Z1...</a>	16,252,928	25
<a href="#">sharedBankConflicts.cu:95 (0x7fc37325d410 in _Z1...</a>	16,252,928	25
<a href="#">sharedBankConflicts.cu:95 (0x7fc37325d400 in _Z1...</a>	16,252,928	25
<a href="#">sharedBankConflicts.cu:88 (0x7fc37325d3e0 in _Z1...</a>	0	0

Overall: 25.00% (16,252,928)  
Contributors: 100.00% derived\_\_memory\_l1\_wavefronts\_shared\_excessive (16,252,928)

We can also check the Memory Workload Analysis section. It shows a hint for Shared Load Bank Conflicts and suggests looking at the Source Counters section for uncoalesced shared loads. The Shared Memory table shows a high count of bank conflicts.

The screenshot shows the NVIDIA Nsight Compute interface. At the top, the 'Apply Rules' button is visible. Below it, the 'Memory Workload Analysis' section provides a detailed analysis of GPU memory resources. A table titled 'Shared Load Bank Conflicts' shows an estimated speedup of 89.06%. Below this, a table lists key performance indicators. At the bottom, a 'Shared Memory' table is displayed, with the 'Bank Conflicts' column highlighted in orange. The 'Bank Conflicts' column shows a value of 65,011,712 for the 'Total' row.

Metric Name	Value	Guidance
l1tex_data_bank_conflicts_pipe_lsu_mem_shared_op_id sum	6.50117e+07	Decrease bank conflicts for shared loads
l1tex_throughput.avg.pct.of.peak.sustained_active	92.4168	The higher the L1/TEX cache throughput the more severe the issue becomes

	Instructions	Requests	Wavefronts	% Peak	Bank Conflicts
Shared Load	2,097,152	2,097,152	67,458,953	82.52	65,011,712
Shared Load Matrix	0	0			
Shared Store	2,097,152	2,097,152	2,097,152	0.64	0
Shared Store From Global Load	0	0	0	0	0
Shared Atomic	0	0	0	0	0
Other	-	-	1,111,317	3.28	0
Total	4,194,304	4,194,304	70,667,422	86.45	65,011,712

Click on the **Apply Rules** button at the top to apply rules so that we can also see the hints at the source line level on the source page. In the Source Counters section table for the metric **L1 Wavefronts Shared Excessive** click on one of the source lines to view the kernel source at which the bottleneck occurs.

### Source page

The CUDA source for the kernel is shown. When opening the Source page from Source Counters section, the Navigation metric is automatically filled in to match, in this case the **L1 Wavefronts Shared Excessive** metric. You can see this by the bolding in the column header. The source line at which the bottleneck occurs is highlighted.

It shows shared memory bank conflicts at line #95:

```
odata[indexOut + i * height] = tile[threadIdx.x][threadIdx.y + i];
```

The screenshot shows the NVIDIA Nsight Compute interface. The top bar displays the application name and menu options. Below the toolbar, the current kernel is identified as 'transposeCoalesced.ncu-rep'. Performance metrics for the current kernel are shown: 539 - \_Z18transposeCoalescedPFS\_ii (2...), 5.59 msecond, 3,144,057 Cycles, 24 Regs, GPU 0 - NVIDIA RTX A2000, 562.52 cycle/usecond, 8.6 CC, and [50154] sharedBankConflicts.

The source code view shows the following code snippet:

```

73 // Coalesced global memory transpose with shared memory bank conflicts
74 __global__ void transposeCoalesced(float* odata, float* idata, int width,
75 {
76     __shared__ float tile[TILE_DIM][TILE_DIM];
77
78     int xIndex = blockIdx.x * TILE_DIM + threadIdx.x;
79     int yIndex = blockIdx.y * TILE_DIM + threadIdx.y;
80     int indexIn = xIndex + yIndex*width;
81
82     xIndex = blockIdx.y * TILE_DIM + threadIdx.x;
83     yIndex = blockIdx.x * TILE_DIM + threadIdx.y;
84     int indexOut = xIndex + yIndex*height;
85
86     for (int i = 0; i < TILE_DIM; i += BLOCK_ROWS)
87     {
88         tile[threadIdx.y + i][threadIdx.x] = idata[indexIn + i * width];
89     }
90
91     __syncthreads();
92
93     for (int i = 0; i < TILE_DIM; i += BLOCK_ROWS)
94     {
95         odata[indexOut + i * height] = tile[threadIdx.x][threadIdx.y + i];

```

Line 95 is highlighted with a yellow warning marker. A pop-up window provides the following details:

Access Operation	Access Size	L1 Wavefronts Shared Excessive
Load(4), Store(4)	32(8)	65811712

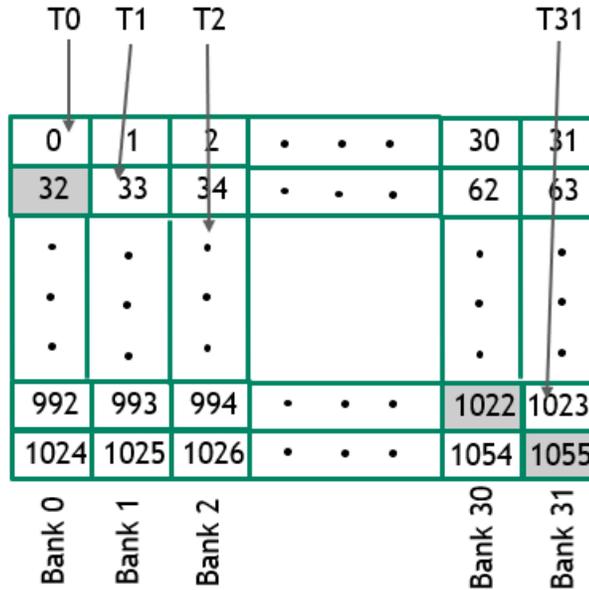
The pop-up window also contains the following text:

⚠ This line is responsible for a high number of warp stalls. See markers on SASS lines for details.

⚠ 96.88% of this line's shared wavefronts are excessive.

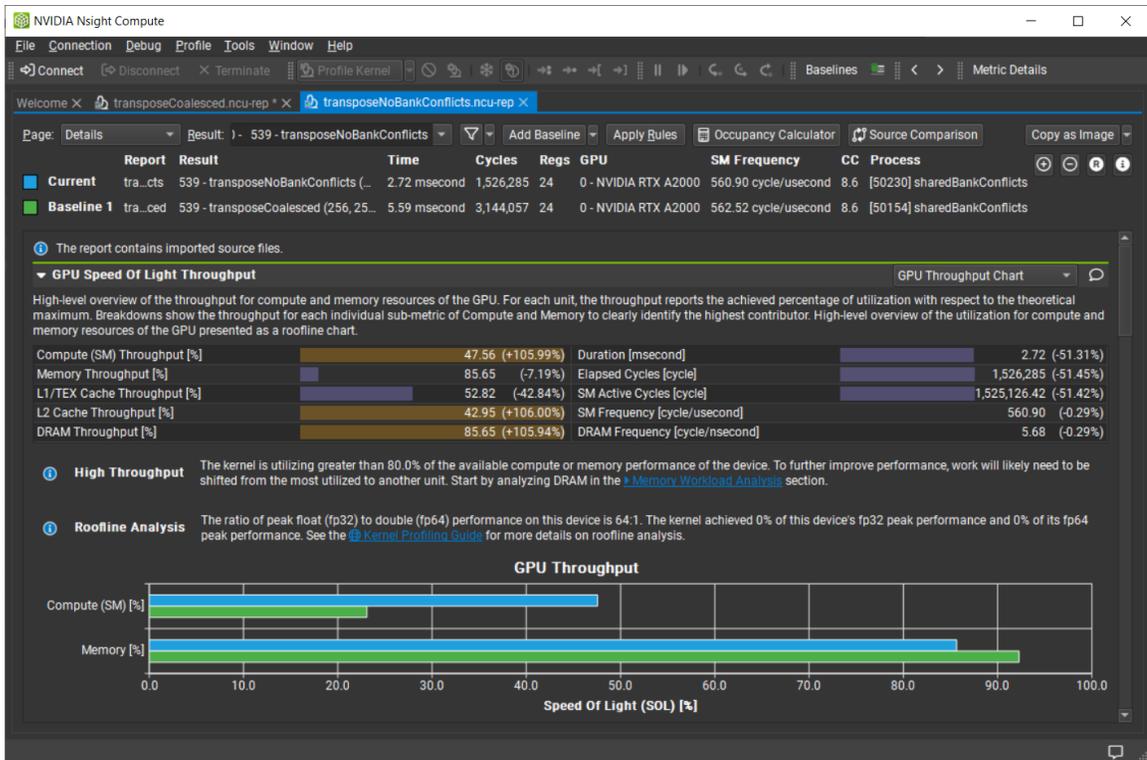
The source page shows notification as Source Markers in the left header of the source code. By hovering the mouse on a marker it shows details in a pop-up window for the specific source line.



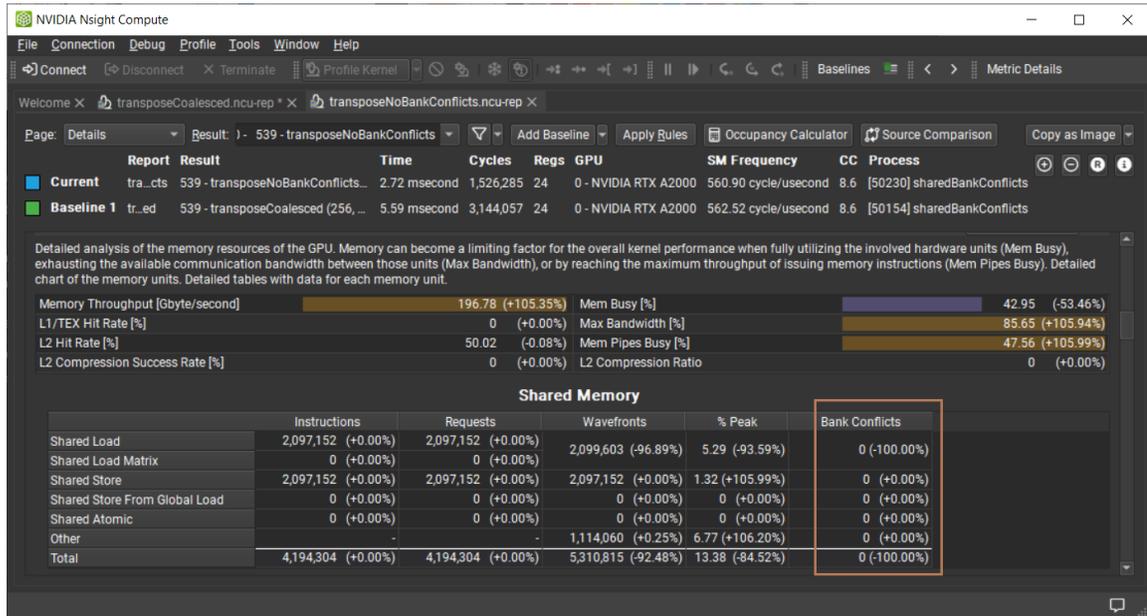


### Profile the updated kernel

The kernel duration has reduced from 5.59 milliseconds to 2.72 milliseconds. We can set a baseline to the initial version of the kernel and compare the profiling results.



We can confirm that there are no shared memory bank conflicts by looking at the Shared Memory metrics table under the Memory workload analysis section.



Note that the reported bank conflicts in the shared memory metrics table under the Memory workload analysis section includes:

- ▶ (A) conflicts within the warp due to shared memory access pattern for the active threads of the warp; and
- ▶ (B) additional conflicts that are caused by multiple clients trying to access the memory banks at the same time, as the L1 Cache and Shared Memory are both backed by the same physical memory banks.

The Source Counters section in the Details page and the Source page only count conflicts of type (A) mentioned above. So in some cases there can be a difference in bank conflict counts between the Memory workload analysis and source counters. Also due to conflicts of type (B) in some cases the bank conflicts can be non-zero for the **transposeNoBankConflicts** kernel in the shared memory table.

# Chapter 6.

## RESOURCES

- ▶ GPU Technology Conference 2022 talk S41723: [How to Understand and Optimize Shared Memory Accesses using Nsight Compute](#)
- ▶ NVIDIA CUDA Sample transpose document - Optimizing Matrix Transpose in CUDA [https://github.com/NVIDIA/cuda-samples/blob/master/Samples/6\\_Performance/transpose/doc/MatrixTranspose.pdf](https://github.com/NVIDIA/cuda-samples/blob/master/Samples/6_Performance/transpose/doc/MatrixTranspose.pdf)
- ▶ NVIDIA CUDA Sample transpose source code [transpose.cu](#)
- ▶ [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/>).