CUDA Performance Optimization

Markus Hrywniak, Senior DevTech Compute, April 2024

Outline

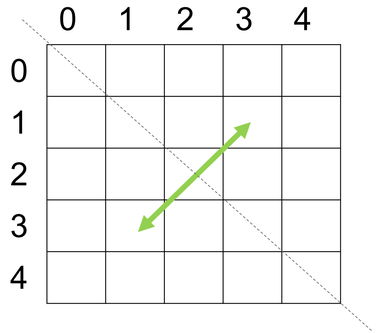
What you will (not) learn this session

- Memory access patterns
  - Coalesced access
- Branch divergence
- Using profiling tools (some more) for Kernel level
- Also important, but not covered here:
  - Exposing enough parallelism
  - Expressing data locality
  - Application-wide profiling

Motivating Example: Matrix Transpose

No FLOPs

CPU VERSION:




```
void transpose(
Integer* a_trans,
Integer* a,
Integer n)
{
  for (Integer row = 0; row < n; ++row)
    for (Integer col = 0; col < n; ++col)
      a_trans[col][row] = a[row][col];
}
```

GPU VERSION:

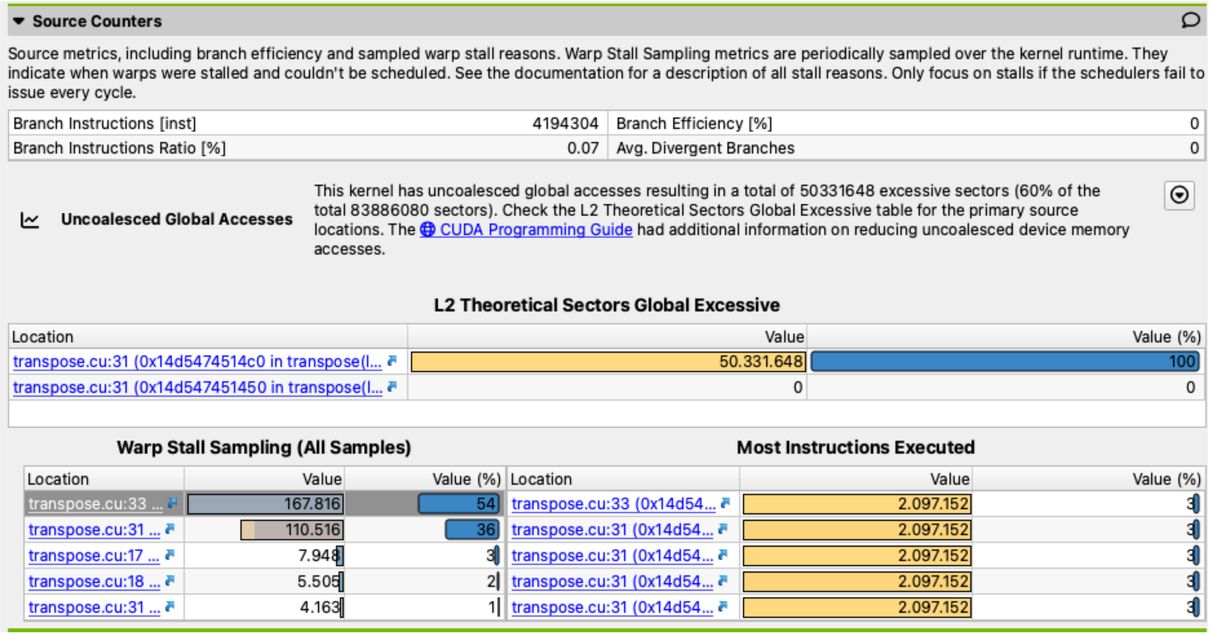
```
__global__ void transpose(
Integer* a_trans,
Integer* a,
Integer n)
{
  Integer row =
        blockIdx.y*blockDim.y+threadIdx.y;
  Integer col =
        blockIdx.x*blockDim.x+threadIdx.x;

  if (row < n && col < n)
    a_trans[col][row] = a[row][col];
}
```

☹

Row-major ordering

Using Nsight Compute (demo)



Source Counters

Source metrics, including branch efficiency and sampled warp stall reasons. Warp Stall Sampling metrics are periodically sampled over the kernel runtime. They indicate when warps were stalled and couldn't be scheduled. See the documentation for a description of all stall reasons. Only focus on stalls if the schedulers fail to issue every cycle.

Branch Instructions [inst]	4194304	Branch Efficiency [%]	0
Branch Instructions Ratio [%]	0.07	Avg. Divergent Branches	0

Uncoalesced Global Accesses

This kernel has uncoalesced global accesses resulting in a total of 50331648 excessive sectors (60% of the total 83886080 sectors). Check the L2 Theoretical Sectors Global Excessive table for the primary source locations. The CUDA Programming Guide had additional information on reducing uncoalesced device memory accesses.

L2 Theoretical Sectors Global Excessive

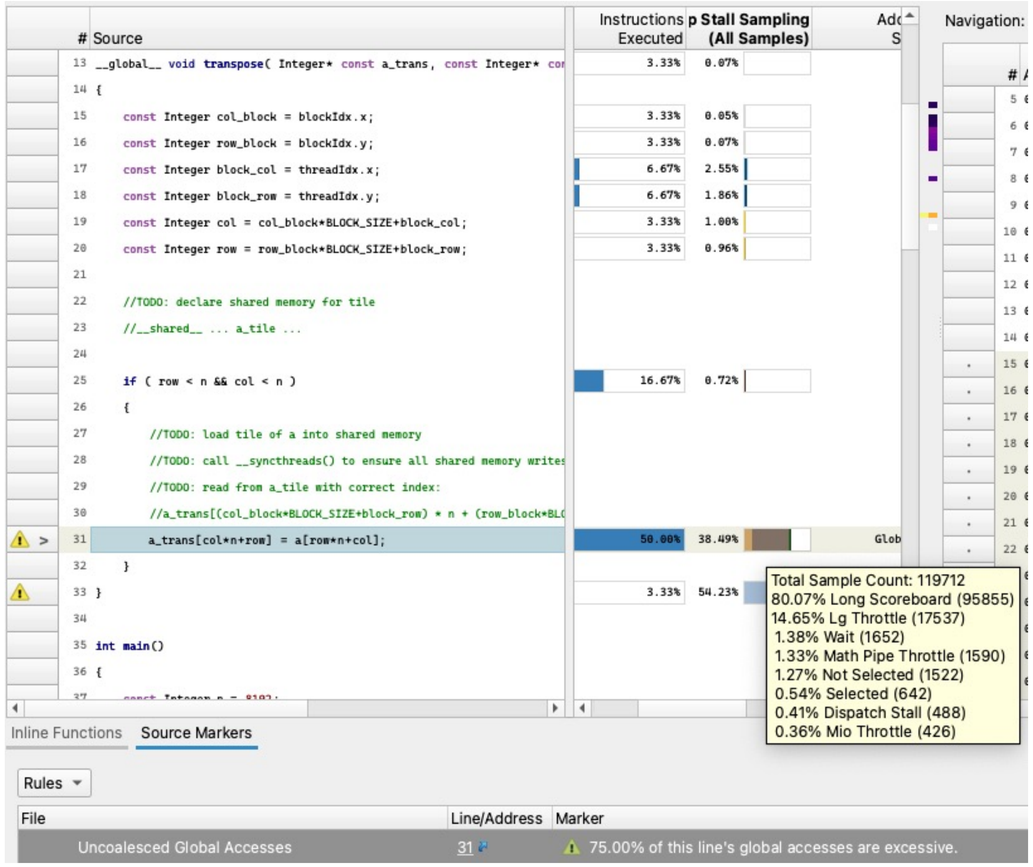
Location	Value	Value (%)
transpose.cu:31 (0x14d5474514c0 in transpose(I...	50.331.648	100
transpose.cu:31 (0x14d547451450 in transpose(I...	0	0

Warp Stall Sampling (All Samples)

Location	Value	Value (%)
transpose.cu:33 ...	167.816	54
transpose.cu:31 ...	110.516	36
transpose.cu:17 ...	7.948	3
transpose.cu:18 ...	5.505	2
transpose.cu:31 ...	4.163	1

Most Instructions Executed

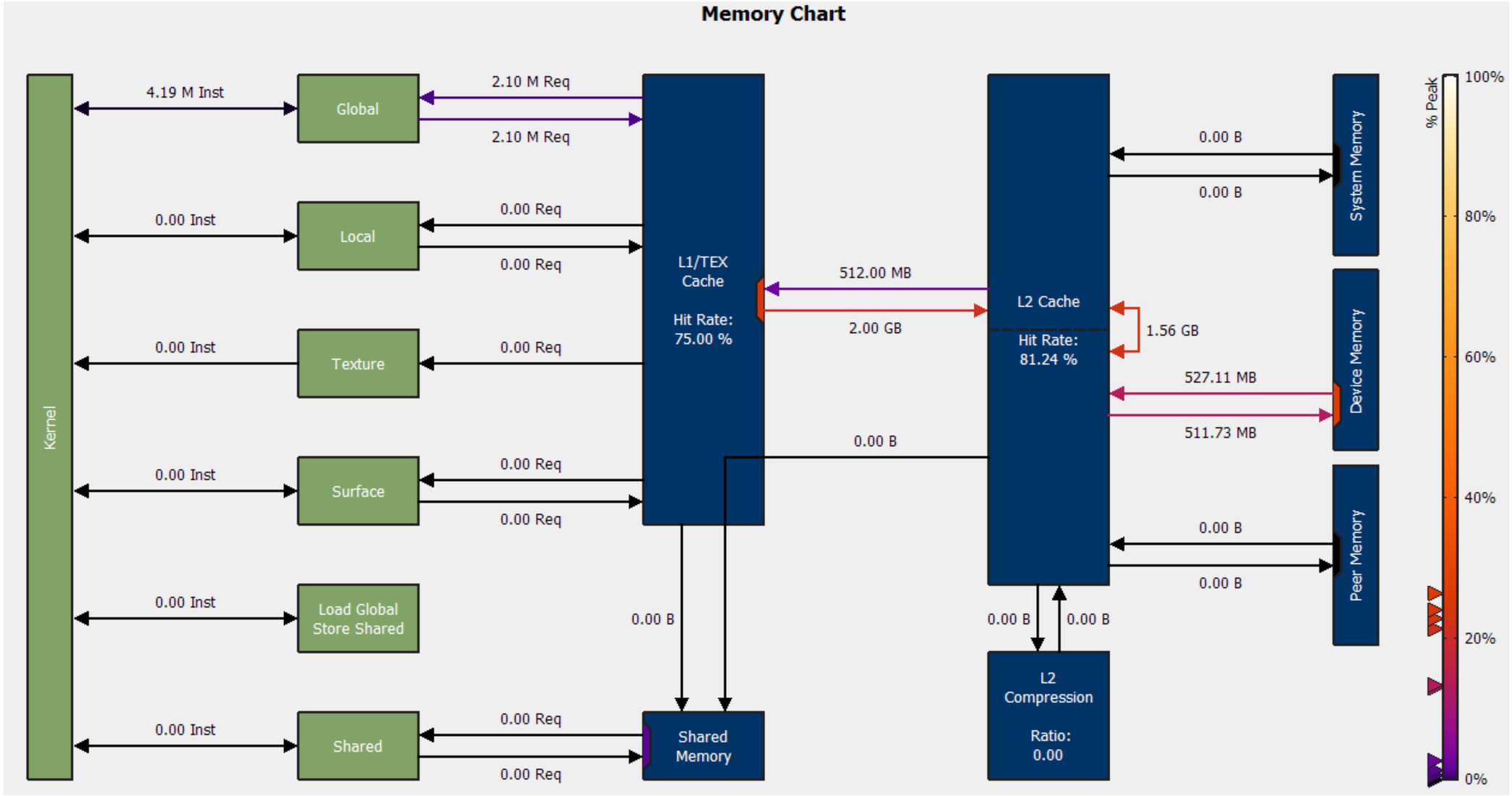
Location	Value	Value (%)
transpose.cu:33 (0x14d54...	2.097.152	3
transpose.cu:31 (0x14d54...	2.097.152	3
transpose.cu:31 (0x14d54...	2.097.152	3
transpose.cu:31 (0x14d54...	2.097.152	3
transpose.cu:31 (0x14d54...	2.097.152	3



„uncoalesced global excesses [...] 60% of the total“

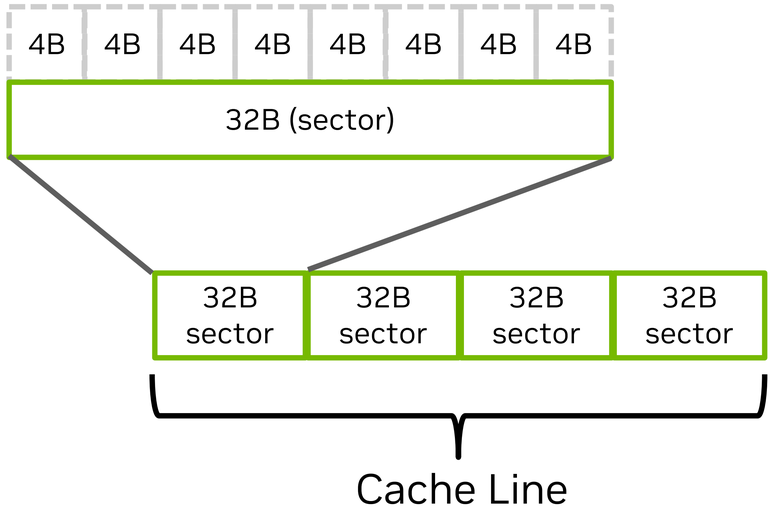
Global, Local, L1, L2?

Understanding the memory hierarchy



Memory Transactions and Coalescing

- Access to global memory triggers transactions (Device Mem Access)
- Memory access granularity = 32 bytes = 1 sector
- Cache line = 128 bytes = 4 consecutive sectors
  - Example: 4 byte per thread → 4B * 32 threads (1 warp) = 128B
- Data goes from **global** device memory through L2 cache
- Granularity*: Sector for L2, Cache line for L1



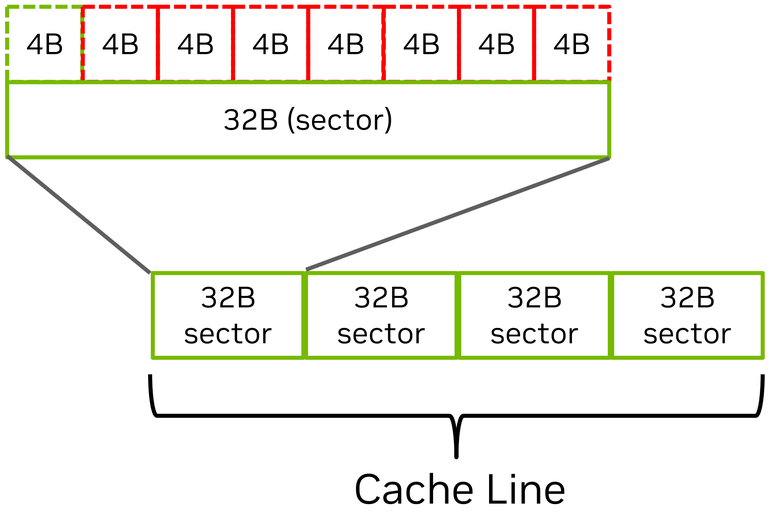
*The full picture:
S32089: Understanding and Optimizing Memory-Bound Kernels with Nsight Compute

Memory Transactions and Coalescing

Coalescing details

- Coalescing: Adjacent accesses can share transactions
- Transactions must be “Naturally Aligned”: First address % size == 0
- All bytes in a transaction are transferred. Use them!

For example, if a 32-byte memory transaction is generated for each thread's 4-byte access, throughput is divided by 8.



$$\text{degree of coalescing} = \frac{\#\text{bytes requested}}{\#\text{bytes transferred}}$$

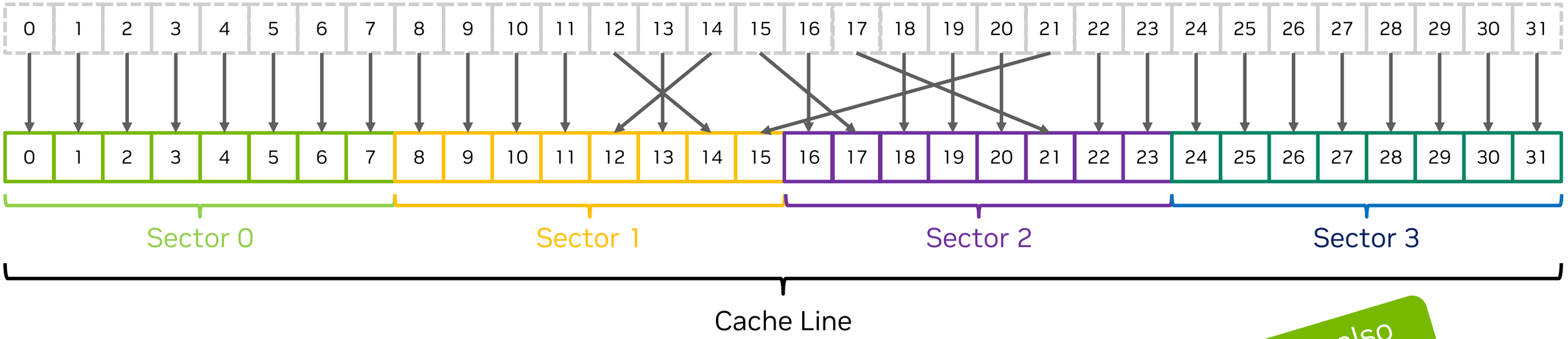
Accessing Global Memory

optimal access pattern (4byte words) – fully coalesced

```
int x_val = x[threadIdx.x];
```

All addresses fall within 4 sectors

Bus utilization: 100%



Permutations are also fine

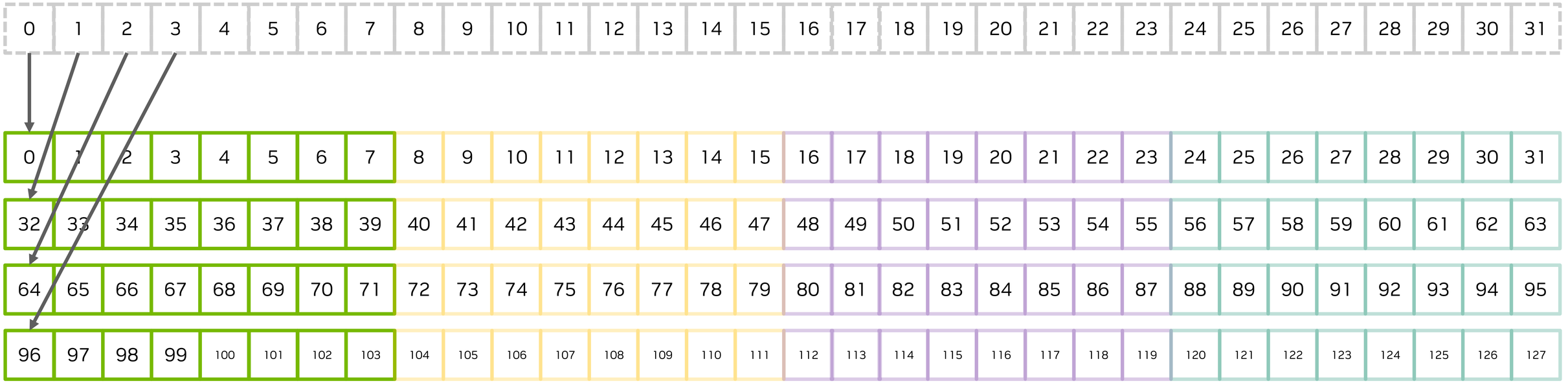
Accessing Global Memory

worst case access pattern (4byte words) – fully uncoalesced

```
// stride 32
int x_val = x[32*threadIdx.x];
// „random“ (pointer chasing, lists, tree, ...)
int x_val = x[lookup[threadIdx.x]];
```

All addresses fall in 32 different sectors

Bus utilization: 12.5%



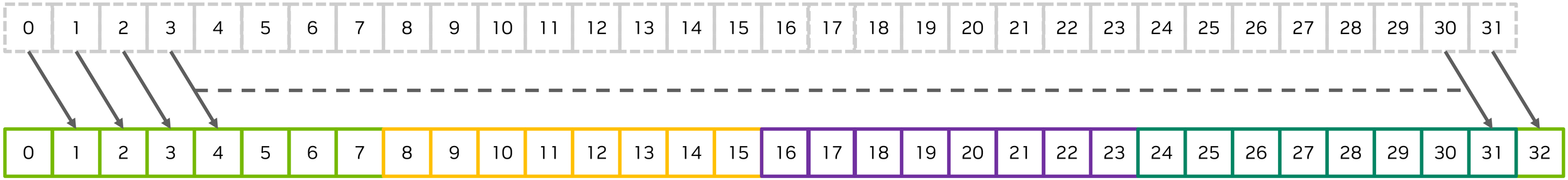
Accessing Global Memory

shifted access

```
int x_val = x[threadIdx.x+1];
```

All addresses fall within 5 sectors

Bus utilization: 80% = 128B/160B



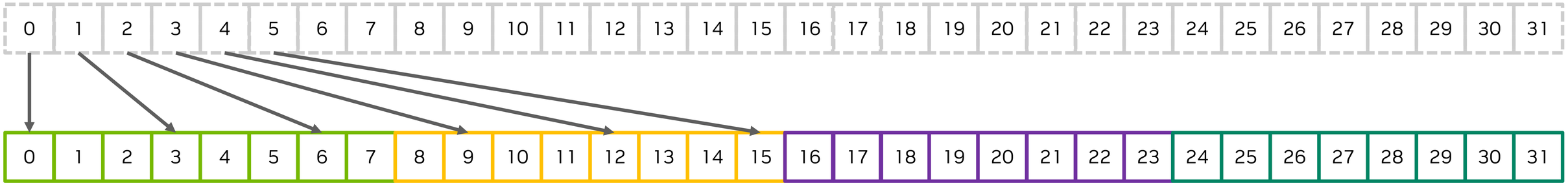
Accessing Global Memory

common access pattern: stride 3

```
int x_val = x[3*threadIdx.x];
```

All addresses fall within 12 sectors (4 byte words)

Bus utilization: 33%



`struct {float x,y,z;} a; ... a[tid].x`

→ use structure-of-arrays (SoA): `a.x[tid]`

`float a[M][N]; ... a[tid][42]`

→ multi-dimensional arrays: pay attention to coalescing (row-major, column-major?)

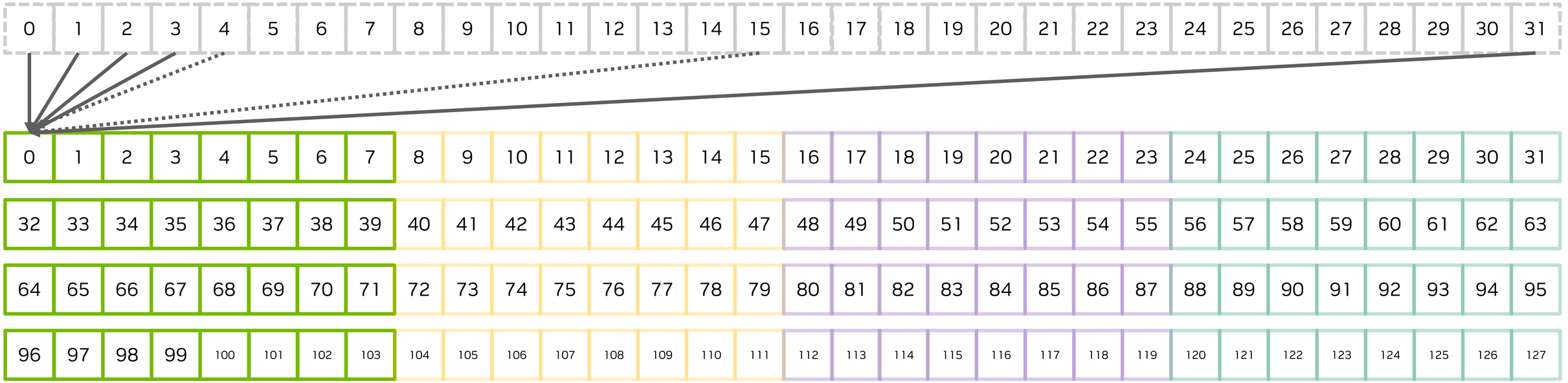
Accessing Global Memory

another “worst case” access pattern?

```
// same for all threads – e.g. loop index
int x_val = x[i];
```

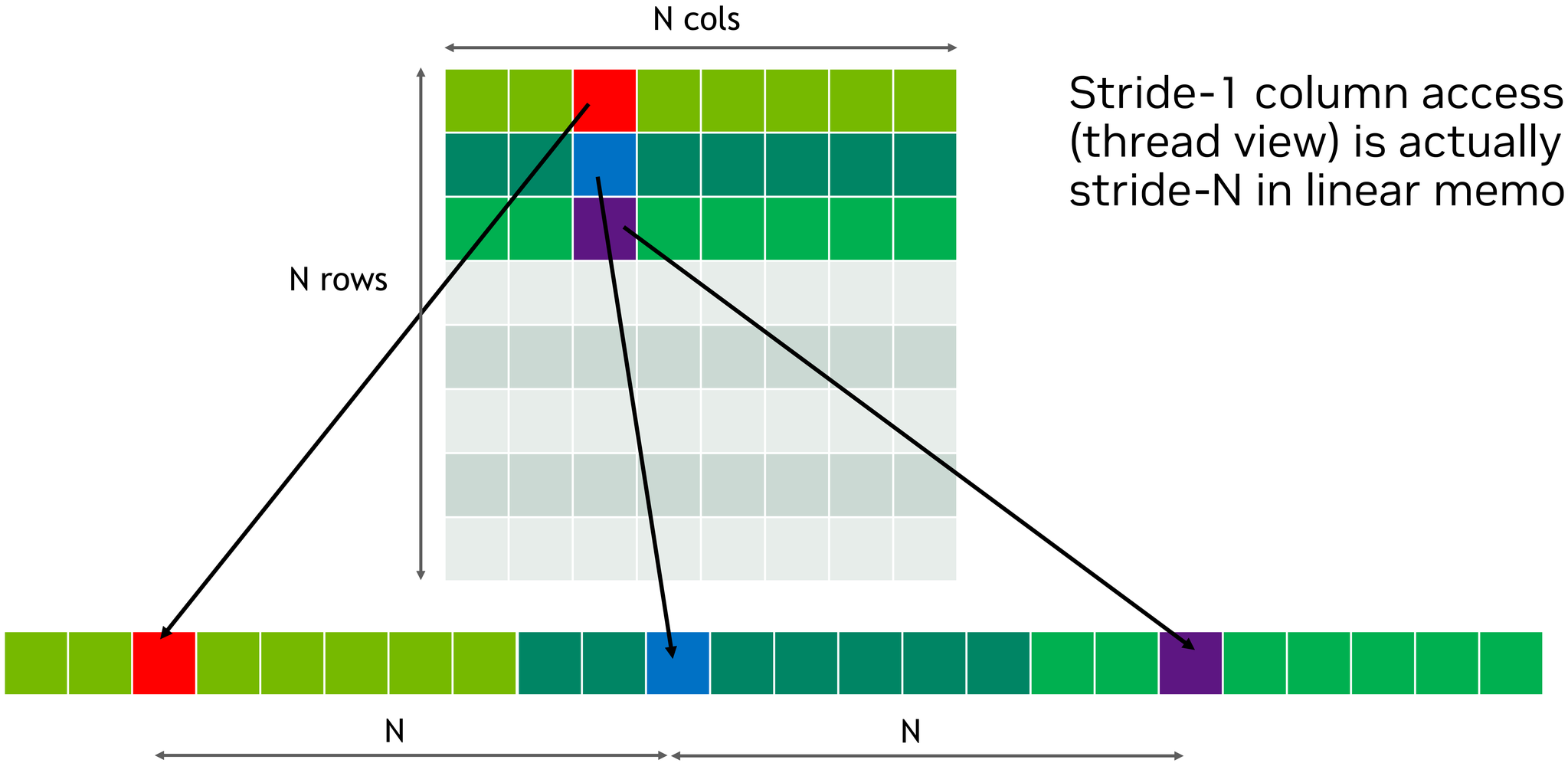
Single address, single sector

Bus utilization: 12.5%



Matrix Transpose

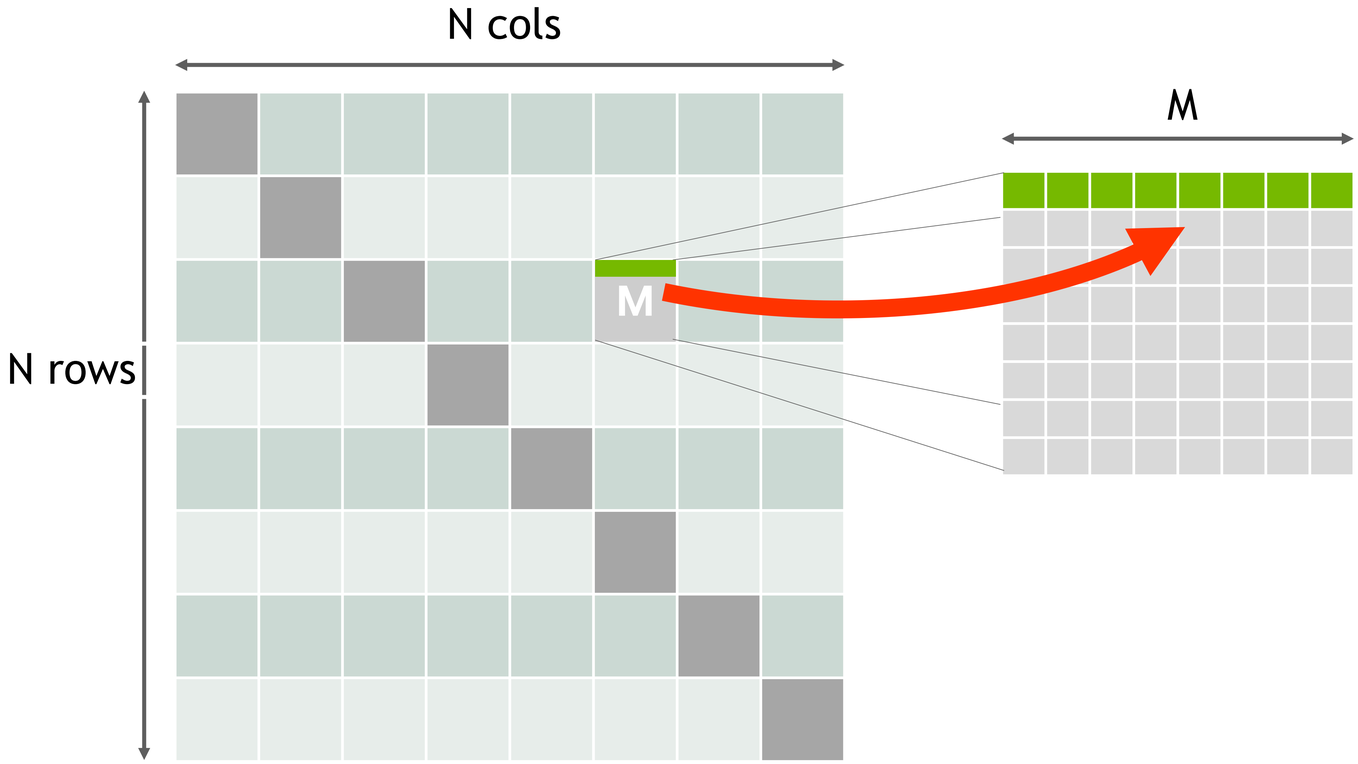
Access pattern



Stride-1 column access (thread view) is actually stride-N in linear memory

Matrix Transpose

Using shared memory

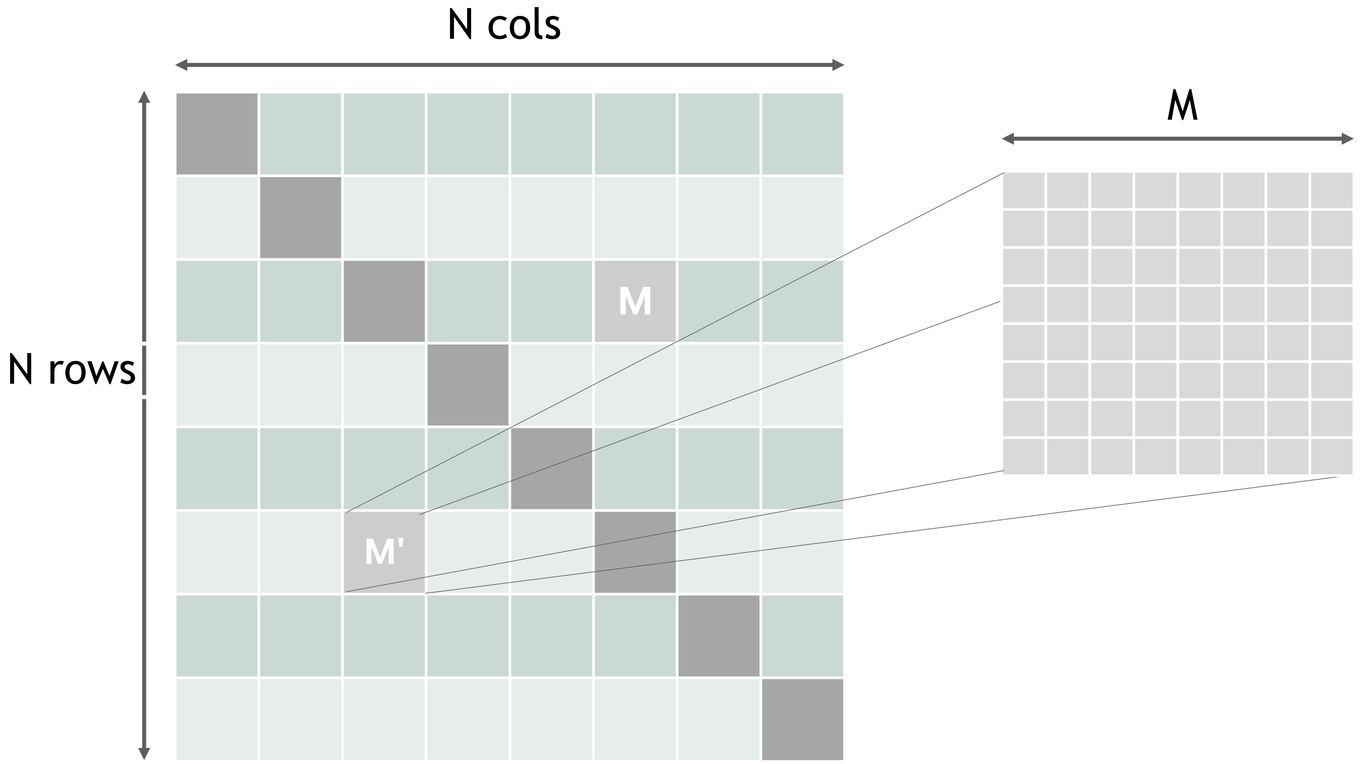


Block row is loaded, fully **coalesced** read

Global Memory **Shared Memory**

Matrix Transpose

Using shared memory

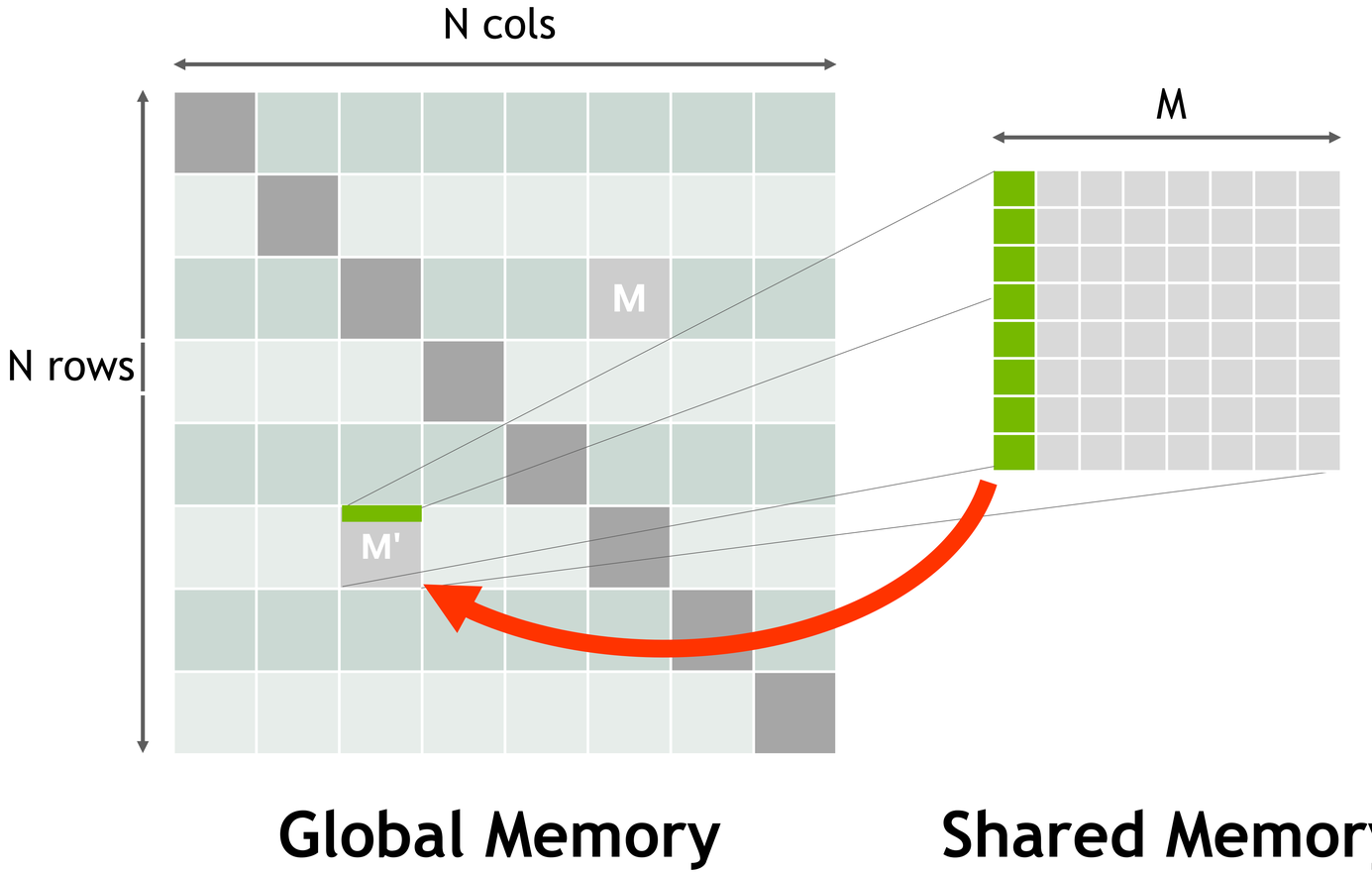


Block row is loaded, fully **coalesced** read

Indexing: Location of block is reflected on the diagonal

Matrix Transpose

Using shared memory



Block row is loaded, fully **coalesced** read

Indexing: Location of block is reflected on the diagonal

Block *column* of shared is written to *row* of matrix

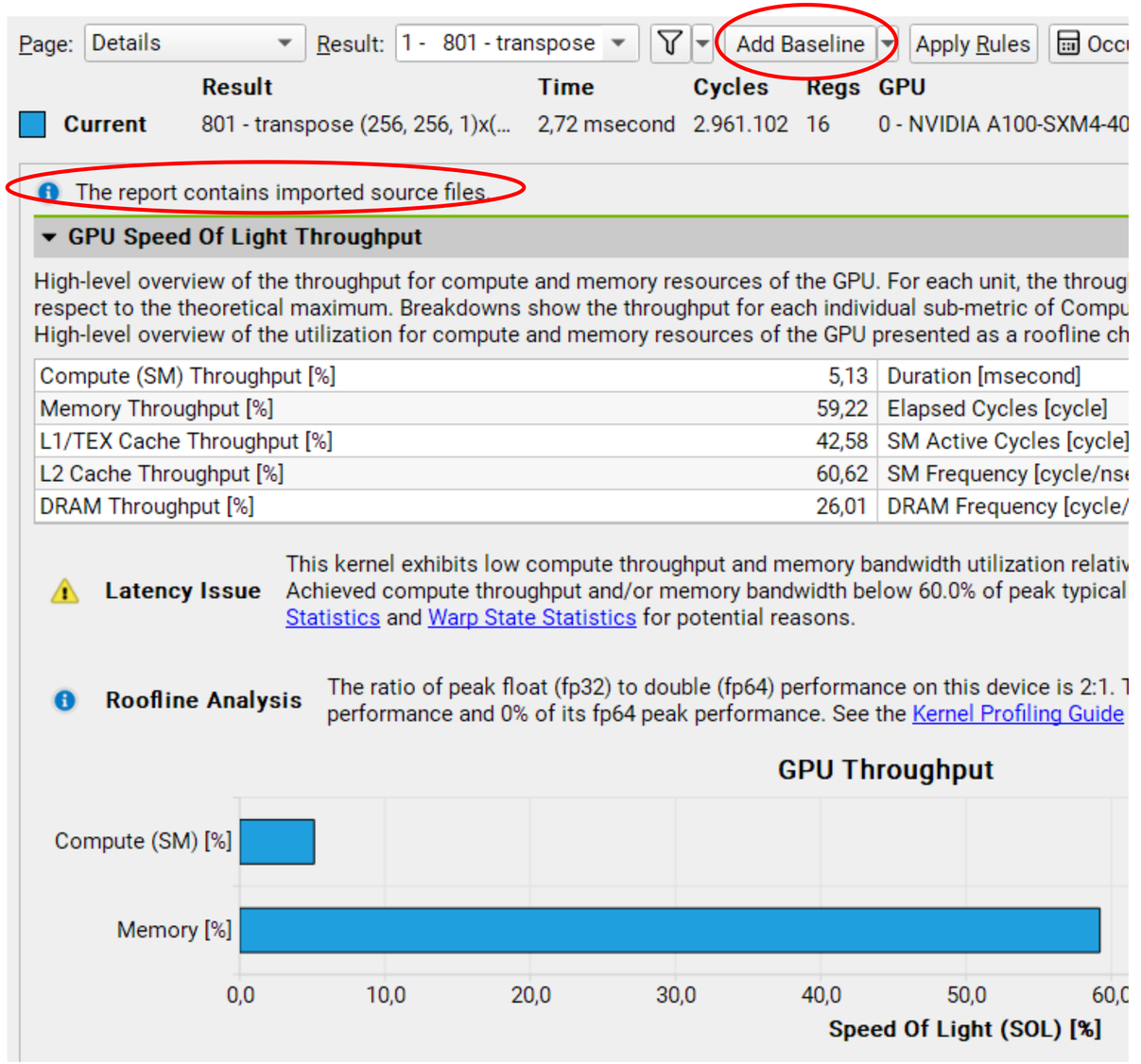
Transposes the block M → M‘

Coalesced write

Analysis with Nsight Compute

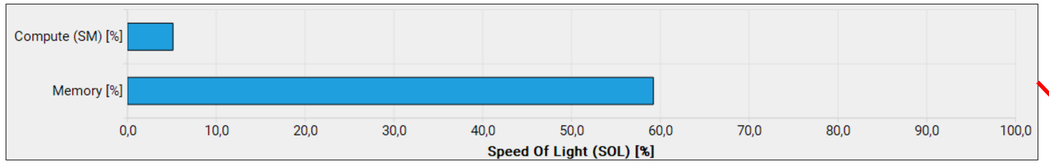
First steps

- Add baseline for comparison
  - Recommended: Always add `--import-source true` if possible
- Check the „Breakdown“ tables and how they change
- Look at the other sections, warp state statistics
- Tooltips on mouseover over metrics/names/...



Kernel-level Profiling

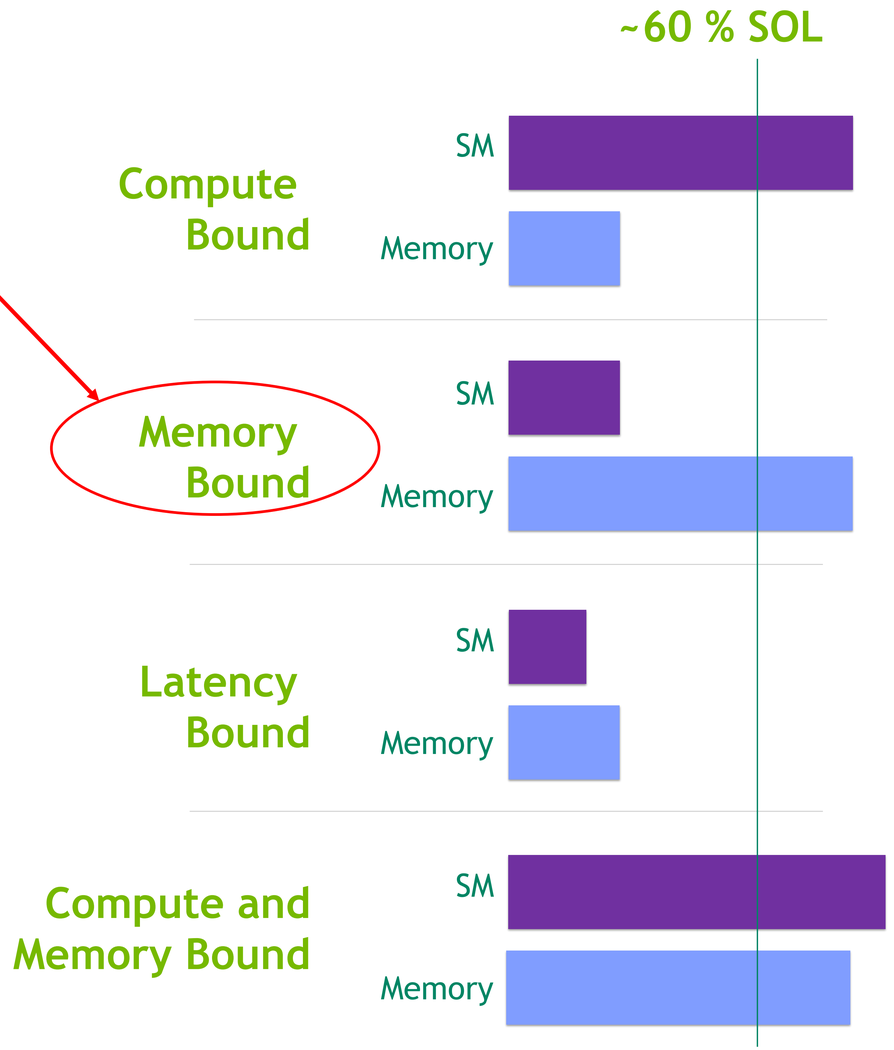
Performance limiter categories



- Four possible combinations of high/low…
  - …memory utilization
  - …compute utilization

- Good? Bad?

→ Depends on problem and its implementation



Analysis with Nsight Compute

Iterating and comparing

- Check the Source Counters section (also on CLI)
- Links will take you to Source/SASS view

Source Counters

Source metrics, including branch efficiency and sampled warp stall reasons. Warp Stall Sampling metrics are periodically sampled over the kernel runtime. They indicate when warps were stalled and couldn't be scheduled. See the documentation for a description of all stall reasons. Only focus on stalls if the schedulers fail to issue every cycle.

Branch Instructions [inst]	4194304	Branch Efficiency [%]	0
Branch Instructions Ratio [%]	0.07	Avg. Divergent Branches	0

Uncoalesced Global Accesses

This kernel has uncoalesced global accesses resulting in a total of 50331648 excessive sectors (60% of the total 83886080 sectors). Check the L2 Theoretical Sectors Global Excessive table for the primary source locations. The CUDA Programming Guide had additional information on reducing uncoalesced device memory accesses.

L2 Theoretical Sectors Global Excessive

Location	Value	Value (%)
transpose.cu:31 (0x14d5474514c0 in transpose(long long...	50.331.648	100
transpose.cu:31 (0x14d547451450 in transpose(long long...	0	0

Warp Stall Sampling (All Samples)

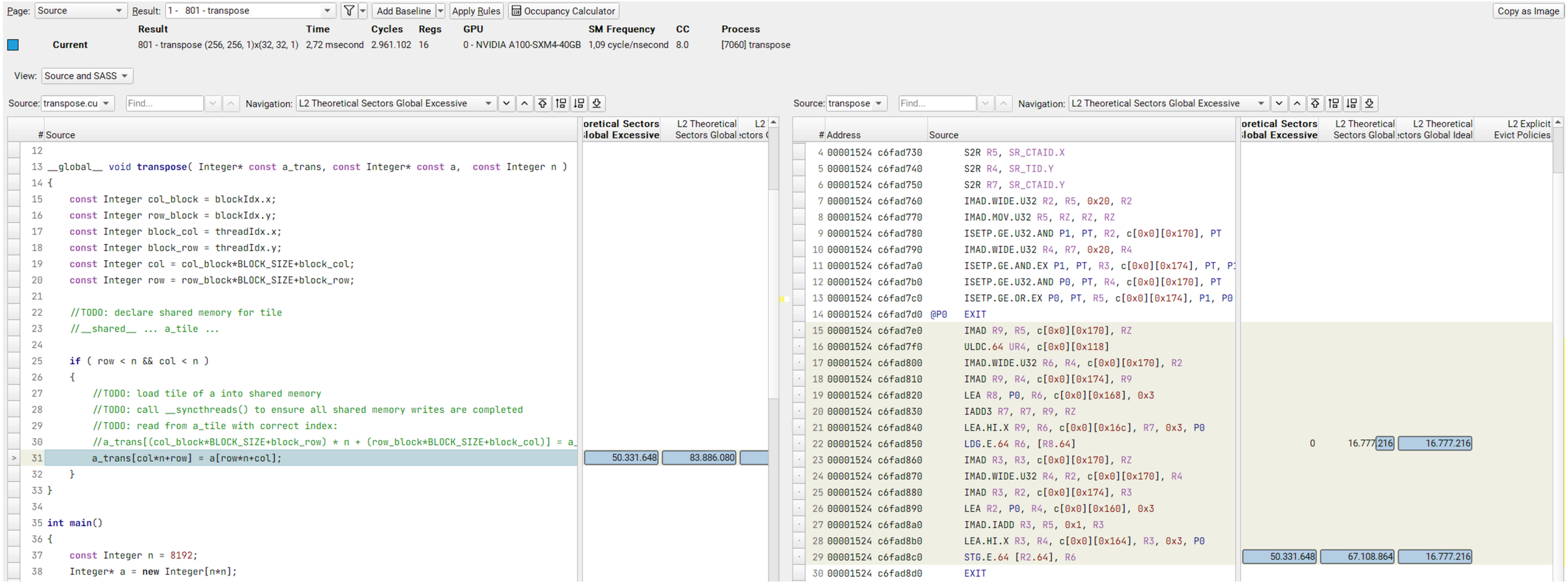
Location	Value	Value (%)
transpose.cu:33 (0x14d5...	167.816	54
transpose.cu:31 (0x14d5...	110.516	36
transpose.cu:17 (0x14d5...	7.948	3
transpose.cu:18 (0x14d5...	5.505	2
transpose.cu:31 (0x14d5...	4.163	1

Most Instructions Executed

Location	Value	Value (%)
transpose.cu:33 (0x14d54...	2.097.152	3
transpose.cu:31 (0x14d54...	2.097.152	3
transpose.cu:31 (0x14d54...	2.097.152	3
transpose.cu:31 (0x14d54...	2.097.152	3
transpose.cu:31 (0x14d54...	2.097.152	3

Analysis with Nsight Compute

Source and SASS view



Task: Coalesced Matrix Transpose

- Location of code: *4-Performance_Optimization/exercises/tasks*
- Steps (see also Instructions.ipynb):
  - Follow TODOs in *transpose.cu*, use shared memory to coalesce writes to *a_trans*
  - Profile with Nsight Compute and observe your changes

Roofline Analysis

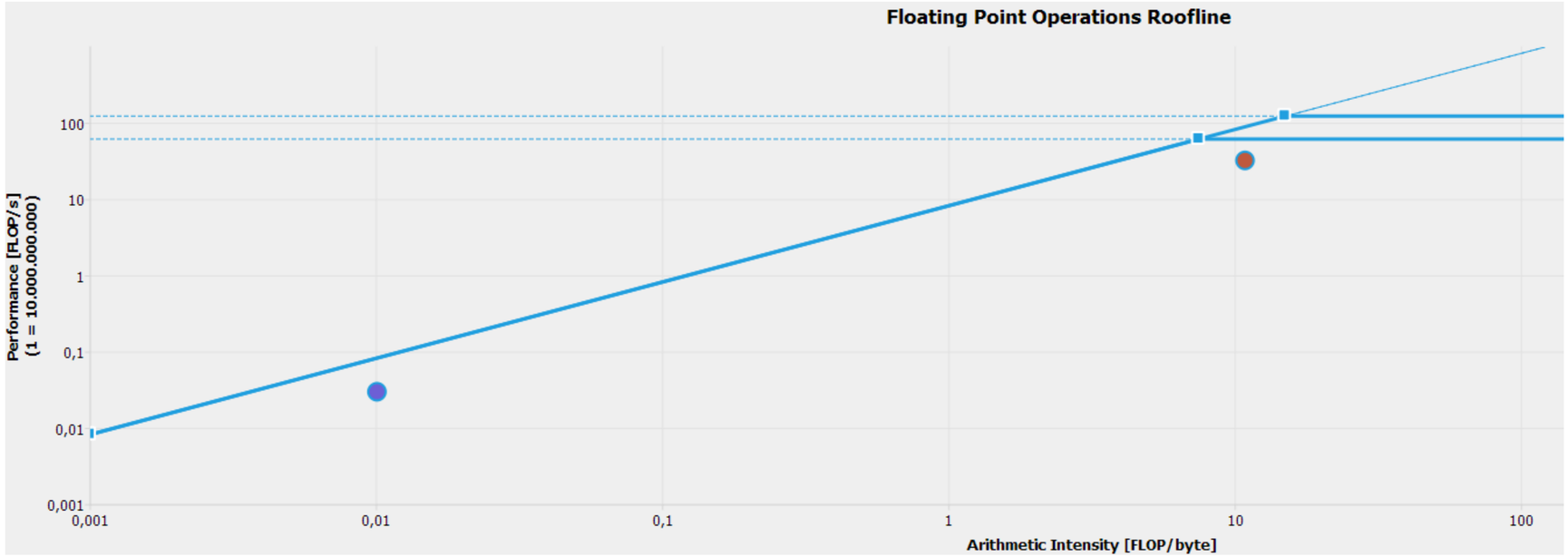
How well is the hardware utilized?

Transpose does zero floating point computations... more interesting example (here: on V100)

Counting flops and transferred bytes → AI, x-axis

Measuring achieved performance →FLOP/s, y-axis

Rooflines from device peak bandwidth / compute



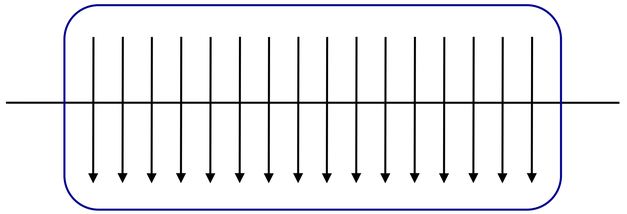
GTC session:
S32062: Performance Tuning CUDA Applications with the Roofline Model

Roofline Hackathon:
https://www.youtube.com/watch?v=ZXZ2SrM3pmE&t=2382s

Branch Divergence

Recap: Warp execution

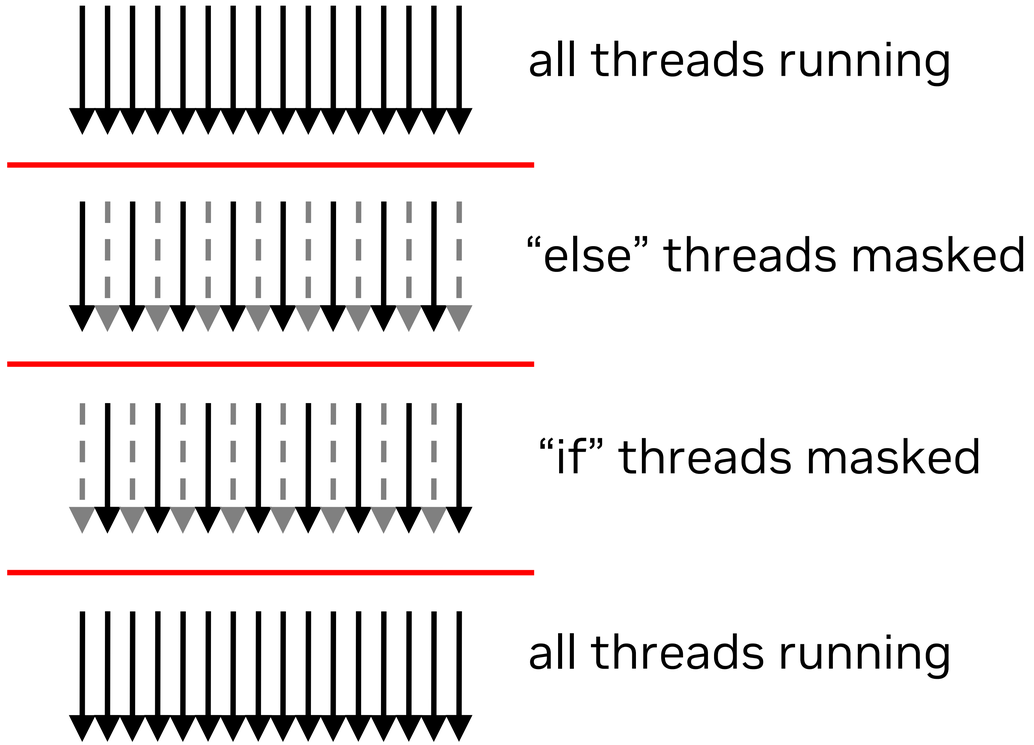
- GPUs use the Single Instruction Multiple Threads (SIMT) execution
  - functionally transparent to the programmer
  - but has performance implications
- warp
  - group of synchronously* executing threads
    (*since Volta: Independent Thread Scheduling)
  - neighbor threads (mostly x dimension)
  - basic unit of scheduling



Branch Divergence

Within Warp

```
// ...
if (condition) {
  // do stuff
  // ...
} else {
  // do other stuff
  // ...
}
// ...
```



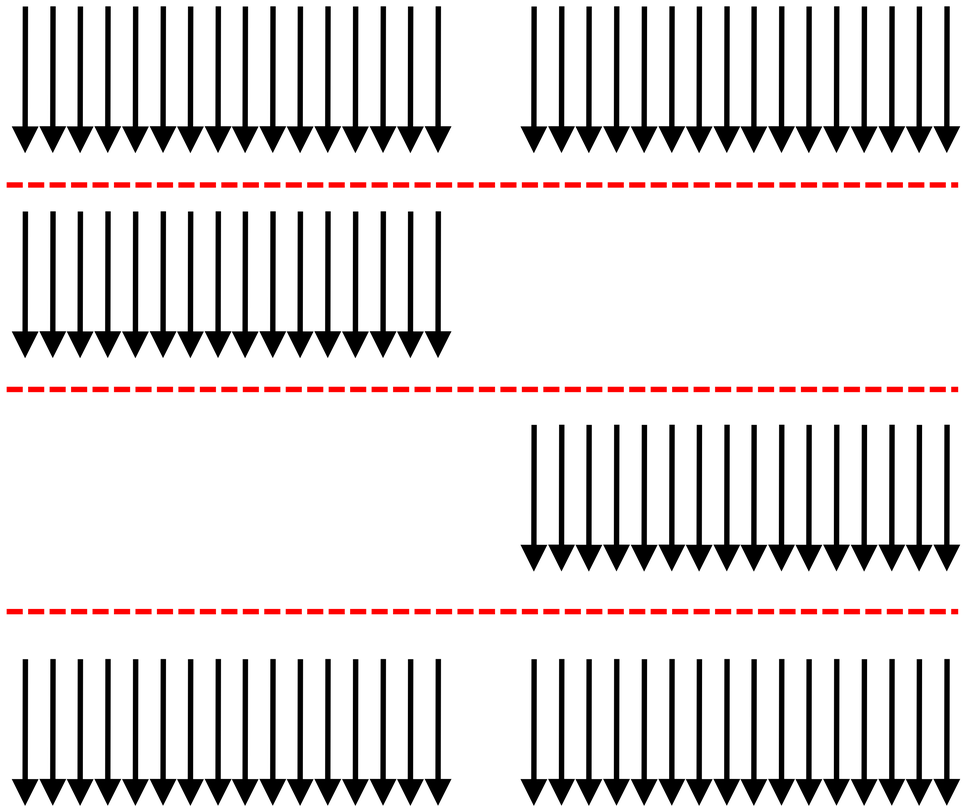
divergence *within* warp → performance penalty

```
if(threadIdx.x % 2 == 0) ...
```

Branch Divergence

Between Warps

```
// ...
if (condition) {
   // do stuff
   // ...
} else {
   // do other stuff
   // ...
}
// ...
```



divergence *between* warps → no penalty

```
if(blockIdx.x % 2 == 0) ...
```

Conclusion

To achieve coalesced global memory access:

- Usually: Fix your access pattern
- Try to use shared memory (but first, check cache behavior)
- Look for different way of storage or better algorithm

Avoid divergent branches

Use the tools!