Sai Jayanth Kalisi
Van Tha Bik Lian

UW - EE 524 GPU
GPU Compute Final Project - Part 2: Design Specification

Focus: CUDA Kernel Tuning Challenge (All Kernels Provided)

PART 1 (3D STENCIL SWEEP)

Stencil patterns are a combination of maps with a local gather over a fixed set of offsets. Every output of a stencil is a function of some neighborhood of elements in the input collection. Its local neighborhood structure exposes opportunities for data usage and optimization of data locality. Inputs are typically divided into a set of partially overlapping strips or regions so that neighbors can be accessed. Outputs are divided into non-overlapping regions so they can be safely written independently in parallel. Stencils are extremely common in image processing and scientific simulations. They are often 2D or 3D (can be extended to higher dimensions), and computationally; they feature high memory traffic, and low arithmetic intensity, and are often memory bound when computation uses all global memory accesses. In this project we study 3D stencils, in particular, we will look at FOUR 3D stencil kernels applied on a given dataset and iteratively fine-tune each kernel to get the best performance.

All our builds, releases, timing results, and profiling will obtained from the AWS AppStream instance with the Turing T4 GPU. We will use NVIDIA's Insight Compute to evaluate our kernels throughout our iterative fine-tuning process. Below are the FOUR kernels we aim to optimize for this part of the project:

ALL THESE KERNELS are 7-POINT 3D Stencil | WE ARE NOT CHANGING STENCIL_RADIUS_3D THROUGHOUT OUR ITERATIVE PROCESS

S1. stencil_3d_basic_kernel
- This is a basic implementation of the 3D sweep stencil kernel using GLOBAL MEMORY ACCESS
- Takes a uniform NxNxN cubic 3D data input grid

S2. stencil_3d_tiled_smem_kernel
- 3D Stencil Sweep with shared memory tiling

S3. stencil_3d_nonsqr_tiled_smem_kernel
- 3D Stencil Sweep with shared memory tiling. Non-sqaure processing for better GMEM coalescing

S4. stencil_3d_thread_coarsening_kernel
- 3D Stencil Sweep - uses shared memory tiling as well as thread coarsening.

PART 2 (HISTOGRAMS)

Histograms display the number count/percentage of occurrences of data values in a dataset and can be used to extract notable features/patterns from large datasets. Histograms can be used to summarize key data distribution characteristics, extract features for object recognition, fraud detection, and so on. Computing each count or bin is an operation that can be done in parallel. In this part of the project we study histograms, in particular, we will look at FOUR kernels for histograms, applied on a given dataset, and iteratively fine-tune each kernel to get the best performance.

All our builds, releases, timing results, and profiling will obtained from the AWS AppStream instance with the Turing T4 GPU. We will use NVIDIA's Insight Compute to evaluate our kernels throughout our iterative fine-tuning process. Below are the FOUR kernels we aim to optimize for this part of the project:

H1. histo_basic_kernel
- Basic GMEM implementation. Uses atomicAdd to accumulate bin values (privatization - takes care of race conditions)

H2. histo_priv_smem_kernel
- SMEM implementation (NO TILING) - also uses atomicAdd (privatization)

H3. histo_coarsen_interleaved_kernel
- Uses SHARED MEM + coarsening with interleaved partitioning. Uses atomicAdd (privaiation)

H4. histo_coarsen_contiguous_kernel
- Uses SHARED MEM + coarsening with contiguous partitioning (each segment assigned to a thread). Uses atomicAdd (privaiation)

PART 3 (Reduction)

Reduction of an array is useful for summarizing all values of that array into a single value and for example, finding the mean, max, min, product, median, etc. It is beneficial for statistical analysis of large swaths of data, in machine and deep learning, physics and engineering simulations, rendering of images, and massively parallel algorithms. In particular, the reduction we intend to implement is that of an aggregate sum across all values in the array provided. In

All our builds, releases, timing results, and profiling will obtained from the AWS AppStream instance with the Turing T4 GPU. We will use NVIDIA's Insight Compute to evaluate our kernels throughout our iterative fine-tuning process. Below are the FOUR kernels we aim to optimize for this part of the project:

All kernels are generalized to handle compute beyond a single thread block/SM - __syncthreads() have worked for threads in the same block/SM. However, there is no way to perform barrier sync between blocks. FOR ALL kernels below, we use atomics to sync between blocks and SMS at the L2 or GMEM level.

R1. reduce_sum_poor_gmem_multiblock_kernel
- GMEM implementation of the reduced sum is the multi-block version, which uses atomics (privatization for partial sums).

R2. reduce_sum_improved_gmem_multiblock_kernel
- This kernel has different strides compared to R1 to improve control+mem divergence. Still uses GMEM, and atomics for partial sums

R3. reduce_sum_segmented_multiblock_kernel
- Implements reduce sum with shared memory usage. The first reduction pass uses GMEM then SMEM thereafter. Atomics is still used for partial sums.

R4. reduce_sum_segmented_coarsened_multiblock_kernel
- Mostly the same as R3. Incorporates CFACT for coarsening.

2 & 3. Tuning/Optimization Table - Initial Results

Please see link:
https://docs.google.com/spreadsheets/d/1nPIY7L3BHG6XiCasgFNKrFSTcV0i-WUeaGxuU5gesas/edit?usp=sharing

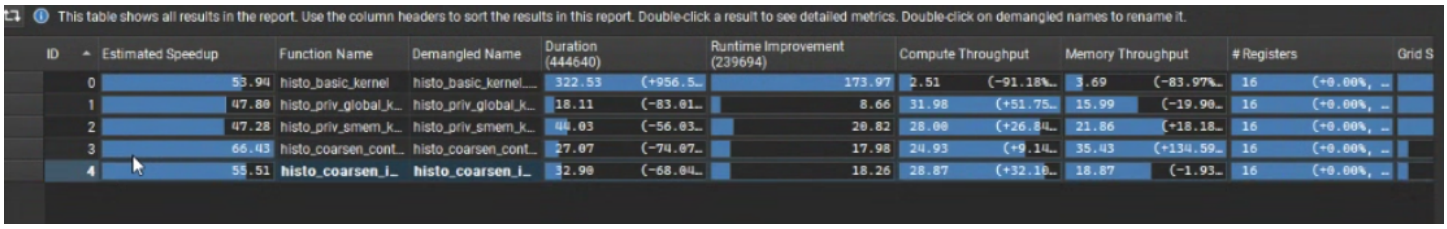
The above link provides a list of launch configurations (including tile sizes, coarsining factors, implemntation types.

There are two tabs in the above link, one for GPU times, and another for CPU times. We hope to fully populate this graph by the end of this project, and show our analysis accordingly.

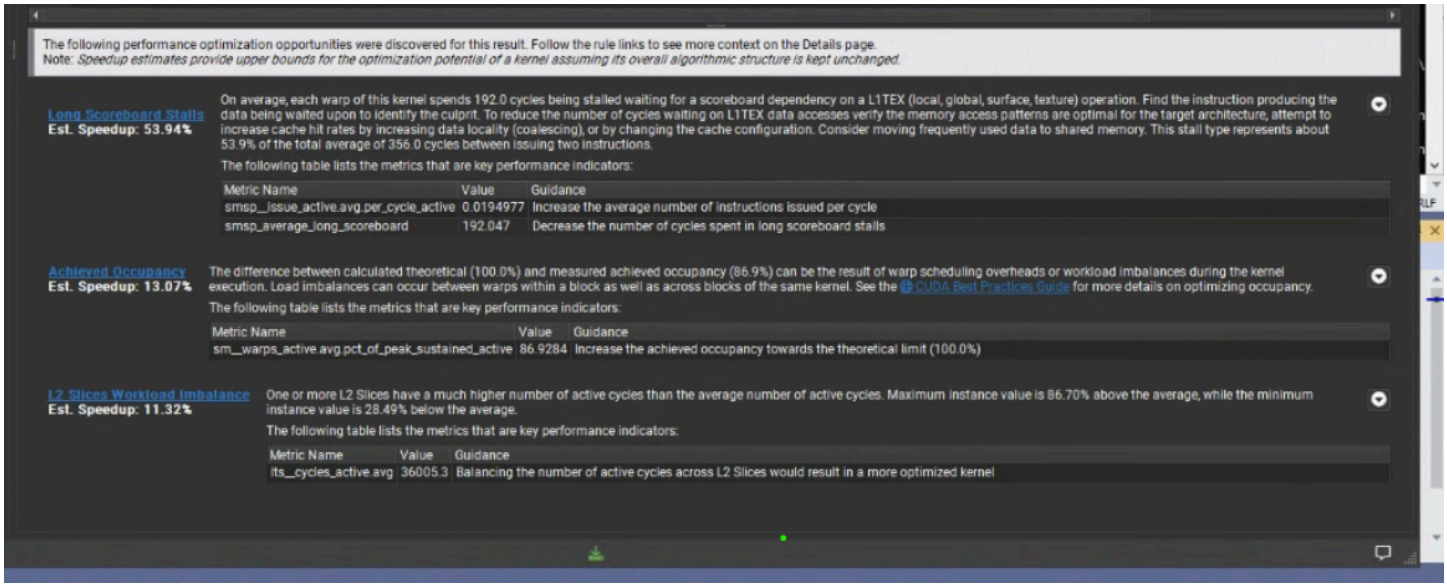
Also, please see the link above for the initial timings and results ran by insight compute. ***Please make sure to see both tabs.***

4. Provide Nsight Compute profiling results for each kernel type set {3D Stencil, Histogram, Reduction}

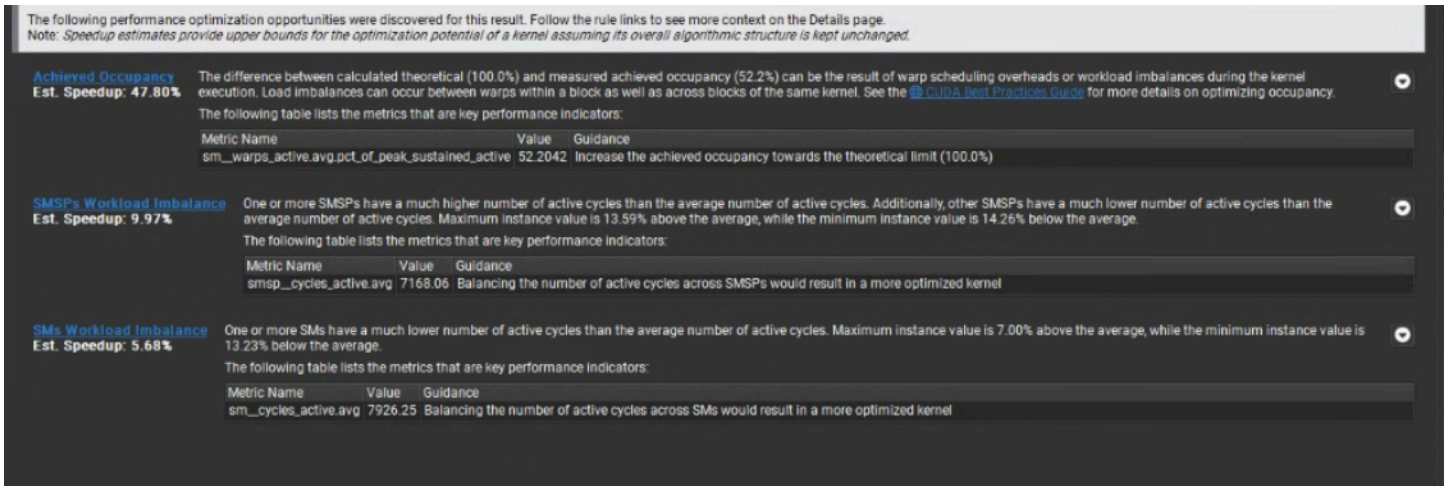
HISTOGRAM



HISTOGRAM: BASIC GMEM KERNEL



HISTOGRAM: PRIVATE GLOBAL KERNEL



The following performance optimization opportunities were discovered for this result. Follow the rule links to see more context on the Details page.
Note: *Speedup estimates provide upper bounds for the optimization potential of a kernel assuming its overall algorithmic structure is kept unchanged.*

Achieved Occupancy
Est. Speedup: 47.80%

The difference between calculated theoretical (100.0%) and measured achieved occupancy (52.2%) can be the result of warp scheduling overheads or workload imbalances during the kernel execution. Load imbalances can occur between warps within a block as well as across blocks of the same kernel. See the CUDA Best Practices Guide for more details on optimizing occupancy.

The following table lists the metrics that are key performance indicators:

Metric Name	Value	Guidance
sm__warps_active.avg.pct_of_peak_sustained_active	52.2042	Increase the achieved occupancy towards the theoretical limit (100.0%)

SMSPs Workload Imbalance
Est. Speedup: 9.97%

One or more SMSPs have a much higher number of active cycles than the average number of active cycles. Additionally, other SMSPs have a much lower number of active cycles than the average number of active cycles. Maximum instance value is 13.59% above the average, while the minimum instance value is 14.26% below the average.

The following table lists the metrics that are key performance indicators:

Metric Name	Value	Guidance
smsp__cycles_active.avg	7168.06	Balancing the number of active cycles across SMSPs would result in a more optimized kernel

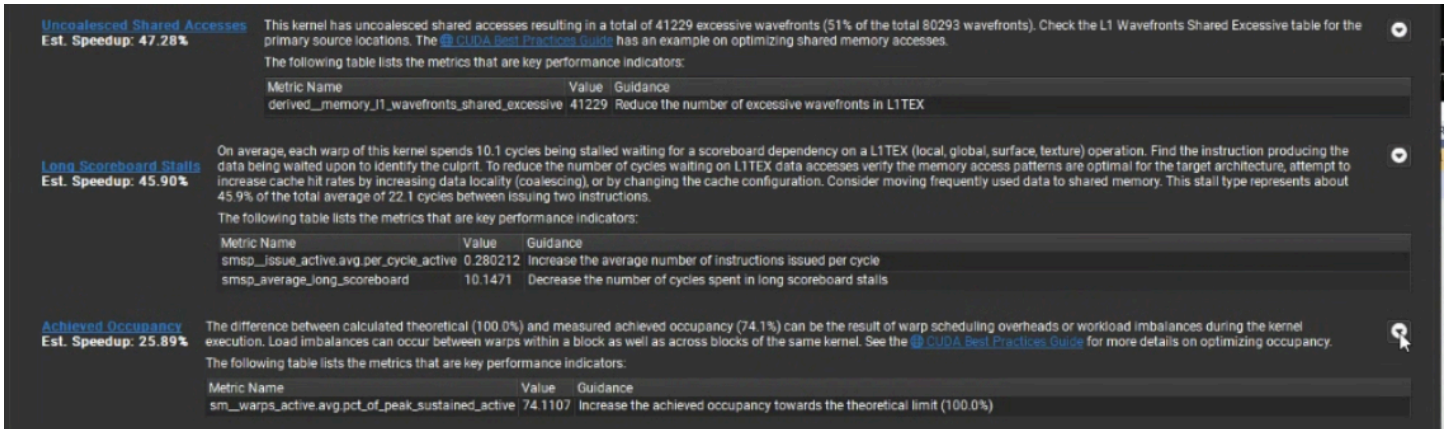
SMs Workload Imbalance
Est. Speedup: 5.68%

One or more SMs have a much lower number of active cycles than the average number of active cycles. Maximum instance value is 7.00% above the average, while the minimum instance value is 13.23% below the average.

The following table lists the metrics that are key performance indicators:

Metric Name	Value	Guidance
sm__cycles_active.avg	7926.25	Balancing the number of active cycles across SMs would result in a more optimized kernel

HISTOGRAM: PRIVATE SMEM KERNEL



Uncoalesced Shared Accesses
Est. Speedup: 47.28%

This kernel has uncoalesced shared accesses resulting in a total of 41229 excessive wavefronts (51% of the total 80293 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.

The following table lists the metrics that are key performance indicators:

Metric Name	Value	Guidance
derived__memory_l1_wavefronts_shared_excessive	41229	Reduce the number of excessive wavefronts in L1TEX

Long Scoreboard Stalls
Est. Speedup: 45.90%

On average, each warp of this kernel spends 10.1 cycles being stalled waiting for a scoreboard dependency on a L1TEX (local, global, surface, texture) operation. Find the instruction producing the data being waited upon to identify the culprit. To reduce the number of cycles waiting on L1TEX data accesses verify the memory access patterns are optimal for the target architecture, attempt to increase cache hit rates by increasing data locality (coalescing), or by changing the cache configuration. Consider moving frequently used data to shared memory. This stall type represents about 45.9% of the total average of 22.1 cycles between issuing two instructions.

The following table lists the metrics that are key performance indicators:

Metric Name	Value	Guidance
smsp__issue_active.avg.per_cycle_active	0.280212	Increase the average number of instructions issued per cycle
smsp_average_long_scoreboard	10.1471	Decrease the number of cycles spent in long scoreboard stalls

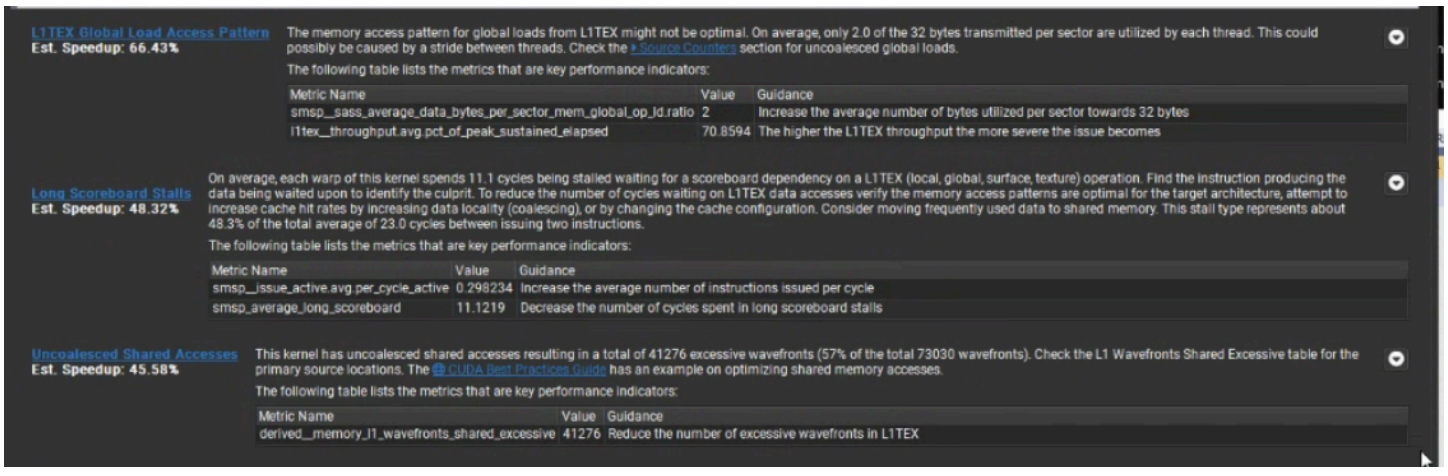
Achieved Occupancy
Est. Speedup: 25.89%

The difference between calculated theoretical (100.0%) and measured achieved occupancy (74.1%) can be the result of warp scheduling overheads or workload imbalances during the kernel execution. Load imbalances can occur between warps within a block as well as across blocks of the same kernel. See the CUDA Best Practices Guide for more details on optimizing occupancy.

The following table lists the metrics that are key performance indicators:

Metric Name	Value	Guidance
sm__warps_active.avg.pct_of_peak_sustained_active	74.1107	Increase the achieved occupancy towards the theoretical limit (100.0%)

HISTOGRAM: COARSEN CONTIGUOUS KERNEL



L1TEX Global Load Access Pattern
Est. Speedup: 66.43%

The memory access pattern for global loads from L1TEX might not be optimal. On average, only 2.0 of the 32 bytes transmitted per sector are utilized by each thread. This could possibly be caused by a stride between threads. Check the Source Counters section for uncoalesced global loads.

The following table lists the metrics that are key performance indicators:

Metric Name	Value	Guidance
smsp__sass_average_data_bytes_per_sector_mem_global_op_ld.ratio	2	Increase the average number of bytes utilized per sector towards 32 bytes
l1tex__throughput.avg.pct_of_peak_sustained_elapsed	70.8594	The higher the L1TEX throughput the more severe the issue becomes

Long Scoreboard Stalls
Est. Speedup: 48.32%

On average, each warp of this kernel spends 11.1 cycles being stalled waiting for a scoreboard dependency on a L1TEX (local, global, surface, texture) operation. Find the instruction producing the data being waited upon to identify the culprit. To reduce the number of cycles waiting on L1TEX data accesses verify the memory access patterns are optimal for the target architecture, attempt to increase cache hit rates by increasing data locality (coalescing), or by changing the cache configuration. Consider moving frequently used data to shared memory. This stall type represents about 48.3% of the total average of 23.0 cycles between issuing two instructions.

The following table lists the metrics that are key performance indicators:

Metric Name	Value	Guidance
smsp__issue_active.avg.per_cycle_active	0.298234	Increase the average number of instructions issued per cycle
smsp_average_long_scoreboard	11.1219	Decrease the number of cycles spent in long scoreboard stalls

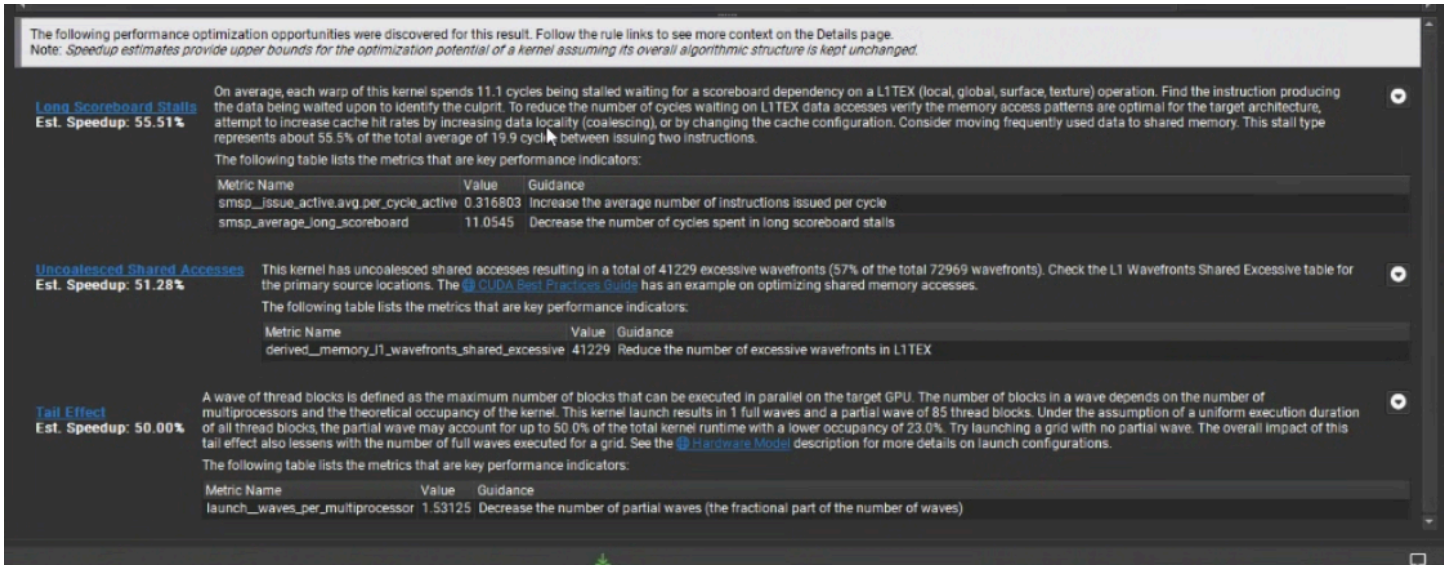
Uncoalesced Shared Accesses
Est. Speedup: 45.58%

This kernel has uncoalesced shared accesses resulting in a total of 41276 excessive wavefronts (57% of the total 73030 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.

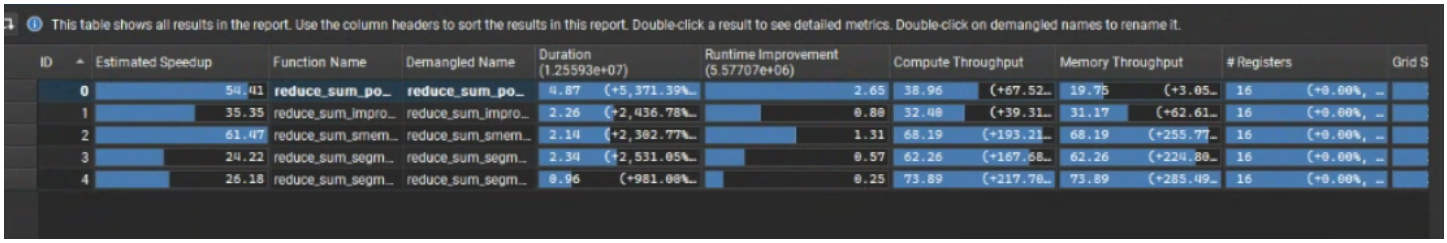
The following table lists the metrics that are key performance indicators:

Metric Name	Value	Guidance
derived__memory_l1_wavefronts_shared_excessive	41276	Reduce the number of excessive wavefronts in L1TEX

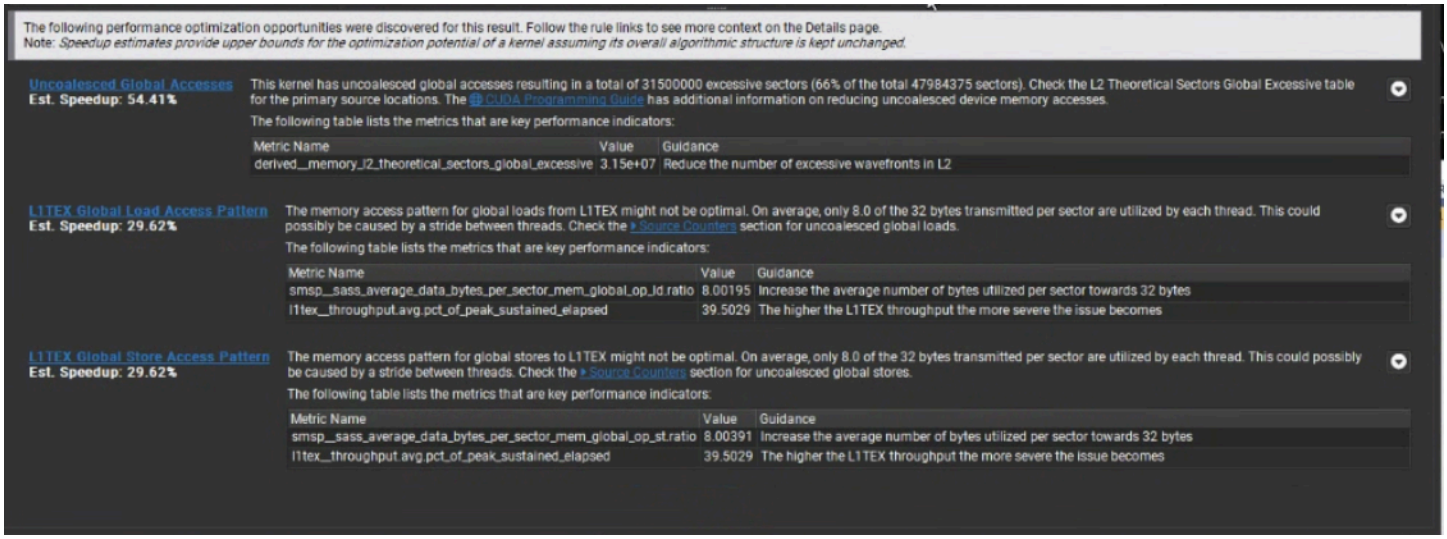
HISTOGRAM: COARSEN INTERLEAVED KERNEL



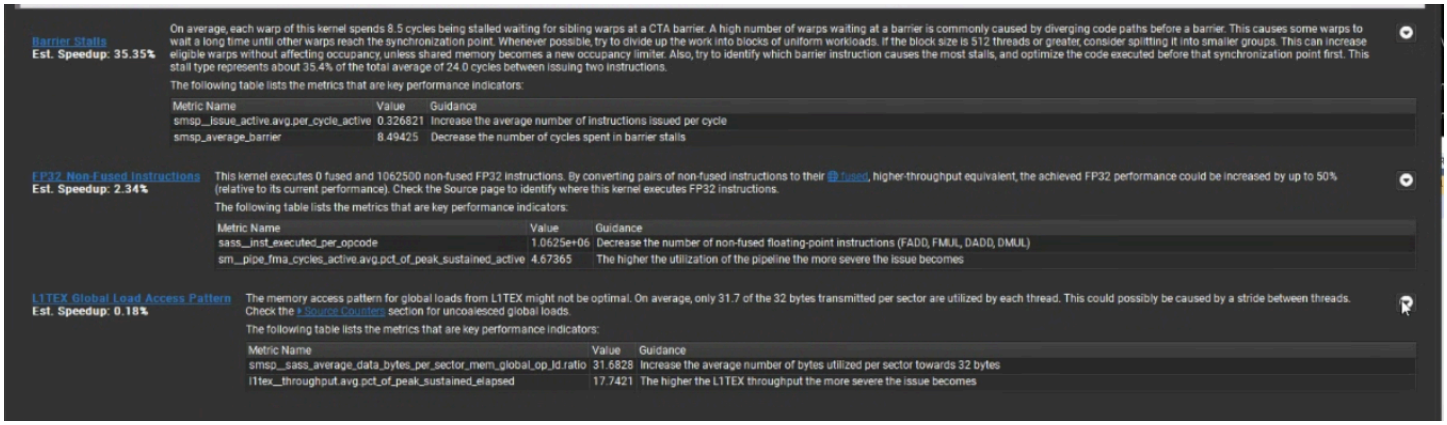
REDUCTION



REDUCTION: POOR GMEM KERNEL (MULTIBLOCK)



REDUCTION: IMPROVED GMEM KERNEL (MULTI BLOCK)



Barrier Stalls
Est. Speedup: 35.35%

On average, each warp of this kernel spends 8.5 cycles being stalled waiting for sibling warps at a CTA barrier. A high number of warps waiting at a barrier is commonly caused by diverging code paths before a barrier. This causes some warps to wait a long time until other warps reach the synchronization point. Whenever possible, try to divide up the work into blocks of uniform workloads. If the block size is 512 threads or greater, consider splitting it into smaller groups. This can increase eligible warps without affecting occupancy, unless shared memory becomes a new occupancy limiter. Also, try to identify which barrier instruction causes the most stalls, and optimize the code executed before that synchronization point first. This stall type represents about 35.4% of the total average of 24.0 cycles between issuing two instructions.

The following table lists the metrics that are key performance indicators:

Metric Name	Value	Guidance
smsp__issue_active.avg.per_cycle_active	0.326821	Increase the average number of instructions issued per cycle
smsp_average_barrier	8.49425	Decrease the number of cycles spent in barrier stalls

FP32 Non-Fused Instructions
Est. Speedup: 2.34%

This kernel executes 0 fused and 1062500 non-fused FP32 instructions. By converting pairs of non-fused instructions to their fused, higher-throughput equivalent, the achieved FP32 performance could be increased by up to 50% (relative to its current performance). Check the Source page to identify where this kernel executes FP32 instructions.

The following table lists the metrics that are key performance indicators:

Metric Name	Value	Guidance
sass__inst_executed_per_opcode	1.0625e+06	Decrease the number of non-fused floating-point instructions (FADD, FMUL, DADD, DMUL)
sm__pipe_fma_cycles_active.avg.pct_of_peak_sustained_active	4.67365	The higher the utilization of the pipeline the more severe the issue becomes

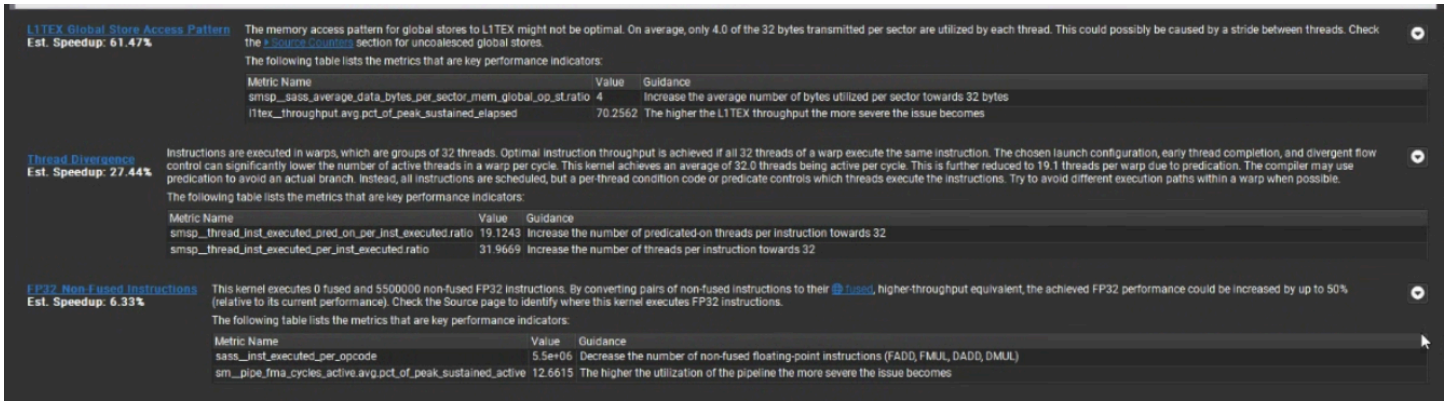
L1TEX Global Load Access Pattern
Est. Speedup: 0.18%

The memory access pattern for global loads from L1TEX might not be optimal. On average, only 31.7 of the 32 bytes transmitted per sector are utilized by each thread. This could possibly be caused by a stride between threads. Check the Source Counters section for uncoalesced global loads.

The following table lists the metrics that are key performance indicators:

Metric Name	Value	Guidance
smsp__sass_average_data_bytes_per_sector_mem_global_op_ld.ratio	31.6828	Increase the average number of bytes utilized per sector towards 32 bytes
l1tex__throughput.avg.pct_of_peak_sustained_elapsed	17.7421	The higher the L1TEX throughput the more severe the issue becomes

REDUCTION: SMEM (SINGLE BLOCK)



L1TEX Global Store Access Pattern
Est. Speedup: 61.47%

The memory access pattern for global stores to L1TEX might not be optimal. On average, only 4.0 of the 32 bytes transmitted per sector are utilized by each thread. This could possibly be caused by a stride between threads. Check the Source Counters section for uncoalesced global stores.

The following table lists the metrics that are key performance indicators:

Metric Name	Value	Guidance
smsp__sass_average_data_bytes_per_sector_mem_global_op_st.ratio	4	Increase the average number of bytes utilized per sector towards 32 bytes
l1tex__throughput.avg.pct_of_peak_sustained_elapsed	70.2562	The higher the L1TEX throughput the more severe the issue becomes

Thread Divergence
Est. Speedup: 27.44%

Instructions are executed in warps, which are groups of 32 threads. Optimal instruction throughput is achieved if all 32 threads of a warp execute the same instruction. The chosen launch configuration, early thread completion, and divergent flow control can significantly lower the number of active threads in a warp per cycle. This kernel achieves an average of 32.0 threads being active per cycle. This is further reduced to 19.1 threads per warp due to predication. The compiler may use predication to avoid an actual branch. Instead, all instructions are scheduled, but a per-thread condition code or predicate controls which threads execute the instructions. Try to avoid different execution paths within a warp when possible.

The following table lists the metrics that are key performance indicators:

Metric Name	Value	Guidance
smsp__thread_inst_executed_pred_on_per_inst_executed.ratio	19.1243	Increase the number of predicated-on threads per instruction towards 32
smsp__thread_inst_executed_per_inst_executed.ratio	31.9669	Increase the number of threads per instruction towards 32

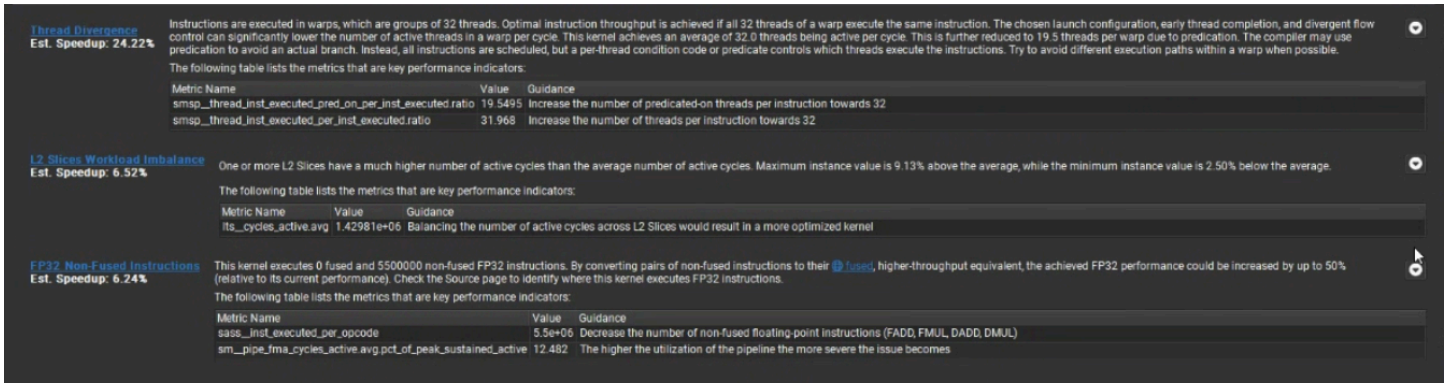
FP32 Non-Fused Instructions
Est. Speedup: 6.33%

This kernel executes 0 fused and 5500000 non-fused FP32 instructions. By converting pairs of non-fused instructions to their fused, higher-throughput equivalent, the achieved FP32 performance could be increased by up to 50% (relative to its current performance). Check the Source page to identify where this kernel executes FP32 instructions.

The following table lists the metrics that are key performance indicators:

Metric Name	Value	Guidance
sass__inst_executed_per_opcode	5.5e+06	Decrease the number of non-fused floating-point instructions (FADD, FMUL, DADD, DMUL)
sm__pipe_fma_cycles_active.avg.pct_of_peak_sustained_active	12.6615	The higher the utilization of the pipeline the more severe the issue becomes

REDUCTION: SEGMENTED, SMEM (MULTI BLOCK)



Thread Divergence
Est. Speedup: 24.22%

Instructions are executed in warps, which are groups of 32 threads. Optimal instruction throughput is achieved if all 32 threads of a warp execute the same instruction. The chosen launch configuration, early thread completion, and divergent flow control can significantly lower the number of active threads in a warp per cycle. This kernel achieves an average of 32.0 threads being active per cycle. This is further reduced to 19.5 threads per warp due to predication. The compiler may use predication to avoid an actual branch. Instead, all instructions are scheduled, but a per-thread condition code or predicate controls which threads execute the instructions. Try to avoid different execution paths within a warp when possible.

The following table lists the metrics that are key performance indicators:

Metric Name	Value	Guidance
smsp__thread_inst_executed_pred_on_per_inst_executed.ratio	19.5495	Increase the number of predicated-on threads per instruction towards 32
smsp__thread_inst_executed_per_inst_executed.ratio	31.968	Increase the number of threads per instruction towards 32

L2 Slices Workload Imbalance
Est. Speedup: 6.52%

One or more L2 Slices have a much higher number of active cycles than the average number of active cycles. Maximum instance value is 9.13% above the average, while the minimum instance value is 2.50% below the average.

The following table lists the metrics that are key performance indicators:

Metric Name	Value	Guidance
lts__cycles_active.avg	1.42981e+06	Balancing the number of active cycles across L2 Slices would result in a more optimized kernel

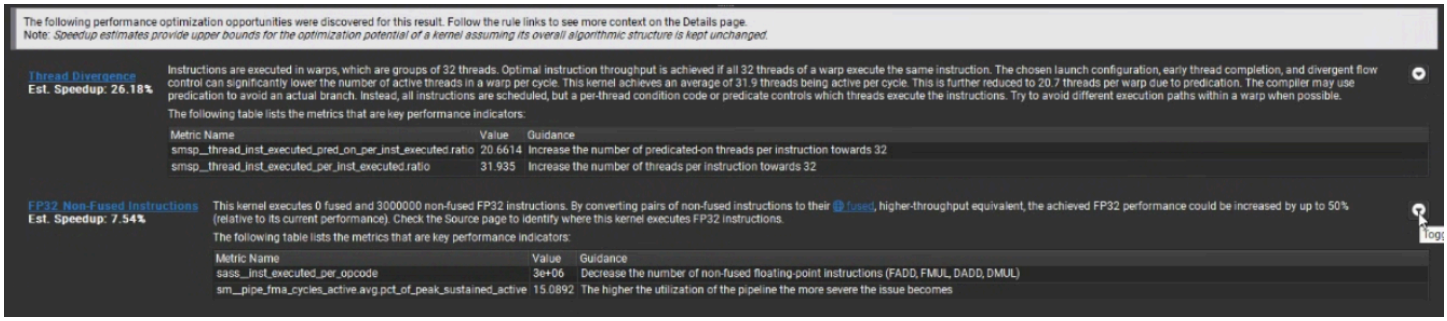
FP32 Non-Fused Instructions
Est. Speedup: 6.24%

This kernel executes 0 fused and 5500000 non-fused FP32 instructions. By converting pairs of non-fused instructions to their fused, higher-throughput equivalent, the achieved FP32 performance could be increased by up to 50% (relative to its current performance). Check the Source page to identify where this kernel executes FP32 instructions.

The following table lists the metrics that are key performance indicators:

Metric Name	Value	Guidance
sass__inst_executed_per_opcode	5.5e+06	Decrease the number of non-fused floating-point instructions (FADD, FMUL, DADD, DMUL)
sm__pipe_fma_cycles_active.avg.pct_of_peak_sustained_active	12.482	The higher the utilization of the pipeline the more severe the issue becomes

REDUCTION: SEGMENTED, COARSENED, SMEM (MULTI BLOCK)



The following performance optimization opportunities were discovered for this result. Follow the rule links to see more context on the Details page.
Note: *Speedup estimates provide upper bounds for the optimization potential of a kernel assuming its overall algorithmic structure is kept unchanged.*

Thread Divergence
Est. Speedup: 26.18%

Instructions are executed in warps, which are groups of 32 threads. Optimal instruction throughput is achieved if all 32 threads of a warp execute the same instruction. The chosen launch configuration, early thread completion, and divergent flow control can significantly lower the number of active threads in a warp per cycle. This kernel achieves an average of 31.9 threads being active per cycle. This is further reduced to 20.7 threads per warp due to predication. The compiler may use predication to avoid an actual branch. Instead, all instructions are scheduled, but a per-thread condition code or predicate controls which threads execute the instructions. Try to avoid different execution paths within a warp when possible.

The following table lists the metrics that are key performance indicators:

Metric Name	Value	Guidance
smsp__thread_inst_executed_pred_on_per_inst_executed.ratio	20.6614	Increase the number of predicated-on threads per instruction towards 32
smsp__thread_inst_executed_per_inst_executed.ratio	31.935	Increase the number of threads per instruction towards 32

FP32 Non-Fused Instructions
Est. Speedup: 7.54%

This kernel executes 0 fused and 3000000 non-fused FP32 instructions. By converting pairs of non-fused instructions to their fused, higher-throughput equivalent, the achieved FP32 performance could be increased by up to 50% (relative to its current performance). Check the Source page to identify where this kernel executes FP32 instructions.

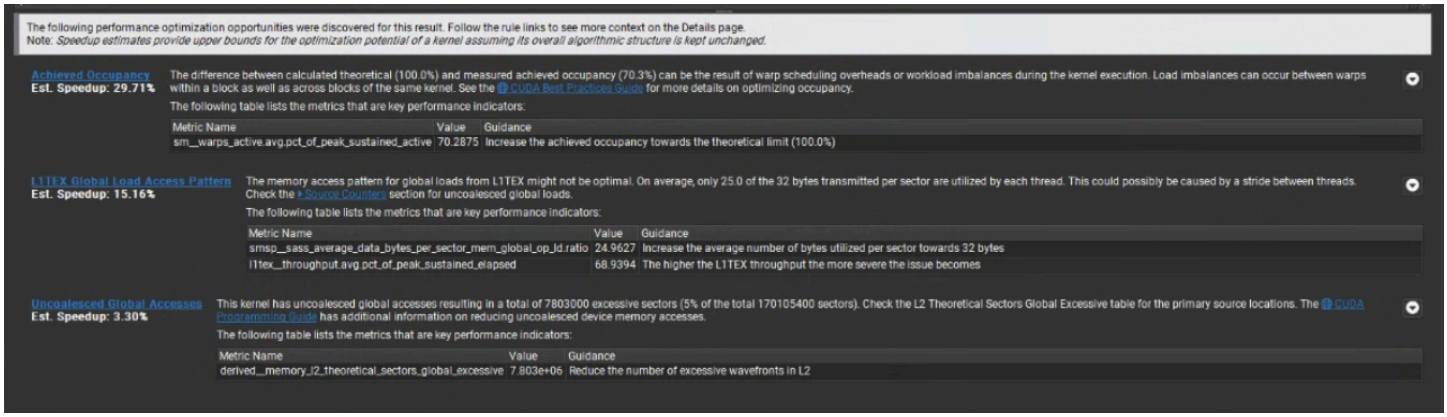
The following table lists the metrics that are key performance indicators:

Metric Name	Value	Guidance
sass__inst_executed_per_opcode	3e+06	Decrease the number of non-fused floating-point instructions (FADD, FMUL, DADD, DMUL)
sm__pipe_fma_cycles_active.avg.pct_of_peak_sustained_active	15.0892	The higher the utilization of the pipeline the more severe the issue becomes

3D STENCIL

ID	Estimated Speedup	Function Name	Demangled Name	(5.38997e+07)	(1.93739e+07)	Compute Throughput	Memory Throughput	# Registers	Grid Size	Block Size
0	29.71	stencil_3d_basi...	stencil_3d_basi...	13.34	3.96	67.25	67.25	24	24, 192, 1...	32, 4, .
1	37.83	stencil_3d_nonsqr...	stencil_3d_nonsqr...	27.68	10.47	49.61	49.61	22	18, 256, 2...	32, 4, .
2	38.34	stencil_3d_thread...	stencil_3d_thread...	12.88	4.94	67.29	67.29	32	18, 18, ...	32, 32, .

3D STENCIL: BASIC GMEM



The following performance optimization opportunities were discovered for this result. Follow the rule links to see more context on the Details page.
Note: *Speedup estimates provide upper bounds for the optimization potential of a kernel assuming its overall algorithmic structure is kept unchanged.*

Achieved Occupancy
Est. Speedup: 29.71%
The difference between calculated theoretical (100.0%) and measured achieved occupancy (70.3%) can be the result of warp scheduling overheads or workload imbalances during the kernel execution. Load imbalances can occur between warps within a block as well as across blocks of the same kernel. See the CUDA Best Practices Guide for more details on optimizing occupancy.
The following table lists the metrics that are key performance indicators:

Metric Name	Value	Guidance
sm__warps_active.avg.pct_of_peak_sustained_active	70.2875	Increase the achieved occupancy towards the theoretical limit (100.0%)

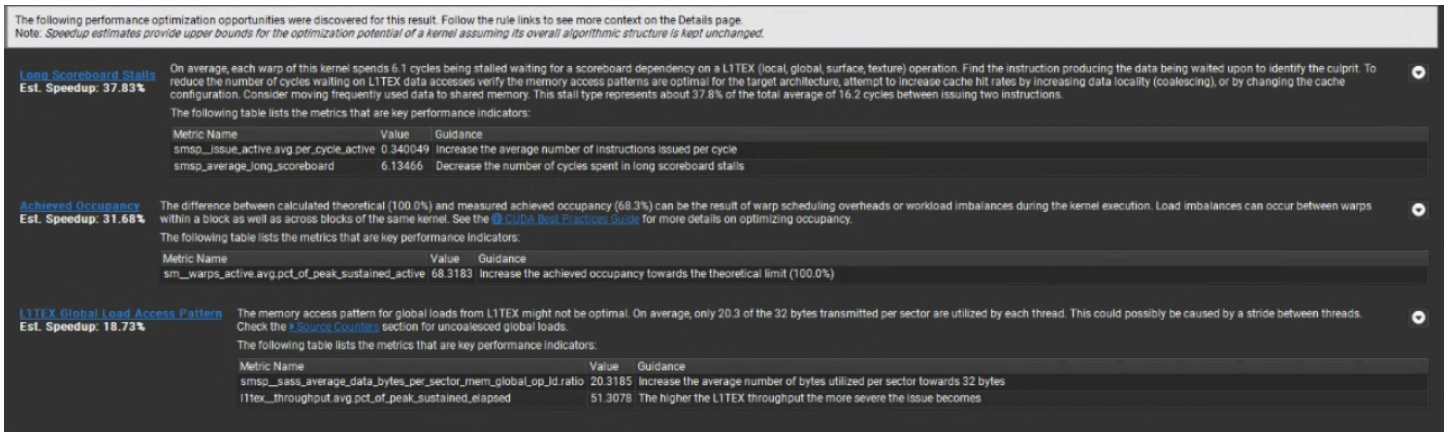
L1TEX Global Load Access Pattern
Est. Speedup: 15.16%
The memory access pattern for global loads from L1TEX might not be optimal. On average, only 25.0 of the 32 bytes transmitted per sector are utilized by each thread. This could possibly be caused by a stride between threads. Check the Source Counters section for uncoalesced global loads.
The following table lists the metrics that are key performance indicators:

Metric Name	Value	Guidance
smsp__sass_average_data_bytes_per_sector_mem_global_op_ld.ratio	24.9627	Increase the average number of bytes utilized per sector towards 32 bytes
l1tex__throughput.avg.pct_of_peak_sustained_elapsed	68.9394	The higher the L1TEX throughput the more severe the issue becomes

Uncoalesced Global Accesses
Est. Speedup: 3.30%
This kernel has uncoalesced global accesses resulting in a total of 7803000 excessive sectors (5% of the total 170105400 sectors). Check the L2 Theoretical Sectors Global Excessive table for the primary source locations. The CUDA Programming Guide has additional information on reducing uncoalesced device memory accesses.
The following table lists the metrics that are key performance indicators:

Metric Name	Value	Guidance
derived__memory_l2_theoretical_sectors_global_excessive	7.803e+06	Reduce the number of excessive wavefronts in L2

3D STENCIL: NONSQUARE SMEM



The following performance optimization opportunities were discovered for this result. Follow the rule links to see more context on the Details page.
Note: *Speedup estimates provide upper bounds for the optimization potential of a kernel assuming its overall algorithmic structure is kept unchanged.*

Long Scoreboard Stalls
Est. Speedup: 37.83%
On average, each warp of this kernel spends 6.1 cycles being stalled waiting for a scoreboard dependency on a L1TEX (local, global, surface, texture) operation. Find the instruction producing the data being waited upon to identify the culprit. To reduce the number of cycles waiting on L1TEX data accesses verify the memory access patterns are optimal for the target architecture, attempt to increase cache hit rates by increasing data locality (coalescing), or by changing the cache configuration. Consider moving frequently used data to shared memory. This stall type represents about 37.8% of the total average of 16.2 cycles between issuing two instructions.
The following table lists the metrics that are key performance indicators:

Metric Name	Value	Guidance
smsp__issue_active.avg.per_cycle_active	0.340049	Increase the average number of instructions issued per cycle
smsp_average_long_scoreboard	6.13466	Decrease the number of cycles spent in long scoreboard stalls

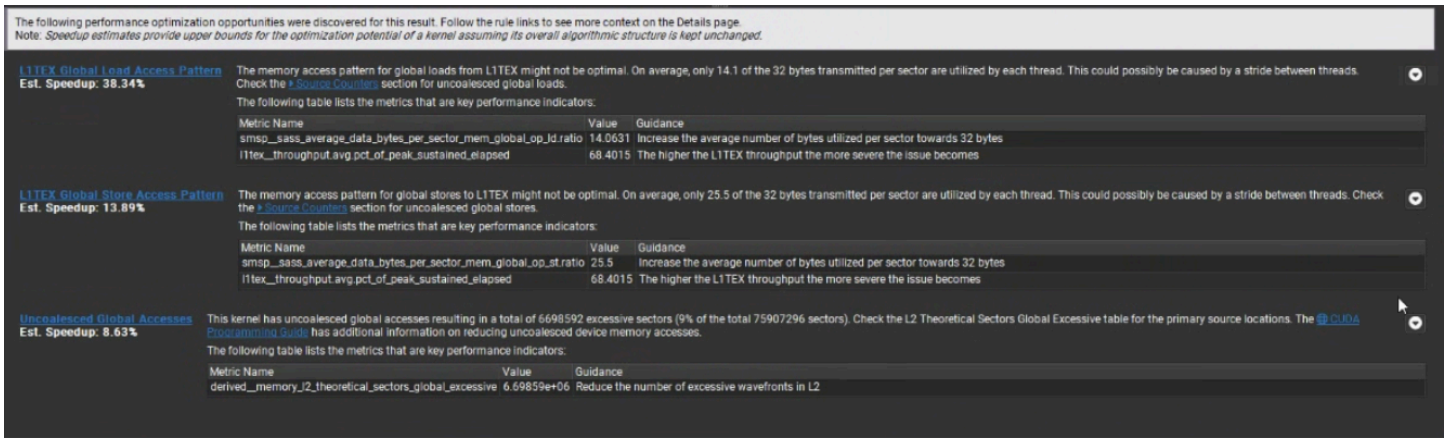
Achieved Occupancy
Est. Speedup: 31.68%
The difference between calculated theoretical (100.0%) and measured achieved occupancy (68.3%) can be the result of warp scheduling overheads or workload imbalances during the kernel execution. Load imbalances can occur between warps within a block as well as across blocks of the same kernel. See the CUDA Best Practices Guide for more details on optimizing occupancy.
The following table lists the metrics that are key performance indicators:

Metric Name	Value	Guidance
sm__warps_active.avg.pct_of_peak_sustained_active	68.3183	Increase the achieved occupancy towards the theoretical limit (100.0%)

L1TEX Global Load Access Pattern
Est. Speedup: 18.73%
The memory access pattern for global loads from L1TEX might not be optimal. On average, only 20.3 of the 32 bytes transmitted per sector are utilized by each thread. This could possibly be caused by a stride between threads. Check the Source Counters section for uncoalesced global loads.
The following table lists the metrics that are key performance indicators:

Metric Name	Value	Guidance
smsp__sass_average_data_bytes_per_sector_mem_global_op_ld.ratio	20.3185	Increase the average number of bytes utilized per sector towards 32 bytes
l1tex__throughput.avg.pct_of_peak_sustained_elapsed	51.3078	The higher the L1TEX throughput the more severe the issue becomes

3D STENCIL: COARSENED, SMEM



The following performance optimization opportunities were discovered for this result. Follow the rule links to see more context on the Details page.
Note: *Speedup estimates provide upper bounds for the optimization potential of a kernel assuming its overall algorithmic structure is kept unchanged.*

L1TEX Global Load Access Pattern
Est. Speedup: 38.34%
The memory access pattern for global loads from L1TEX might not be optimal. On average, only 14.1 of the 32 bytes transmitted per sector are utilized by each thread. This could possibly be caused by a stride between threads. Check the Source Counters section for uncoalesced global loads.
The following table lists the metrics that are key performance indicators:

Metric Name	Value	Guidance
smsp__sass_average_data_bytes_per_sector_mem_global_op_ld.ratio	14.0631	Increase the average number of bytes utilized per sector towards 32 bytes
l1tex__throughput.avg.pct_of_peak_sustained_elapsed	68.4015	The higher the L1TEX throughput the more severe the issue becomes

L1TEX Global Store Access Pattern
Est. Speedup: 13.89%
The memory access pattern for global stores to L1TEX might not be optimal. On average, only 25.5 of the 32 bytes transmitted per sector are utilized by each thread. This could possibly be caused by a stride between threads. Check the Source Counters section for uncoalesced global stores.
The following table lists the metrics that are key performance indicators:

Metric Name	Value	Guidance
smsp__sass_average_data_bytes_per_sector_mem_global_op_st.ratio	25.5	Increase the average number of bytes utilized per sector towards 32 bytes
l1tex__throughput.avg.pct_of_peak_sustained_elapsed	68.4015	The higher the L1TEX throughput the more severe the issue becomes

Uncoalesced Global Accesses
Est. Speedup: 8.63%
This kernel has uncoalesced global accesses resulting in a total of 6698592 excessive sectors (9% of the total 75907296 sectors). Check the L2 Theoretical Sectors Global Excessive table for the primary source locations. The CUDA Programming Guide has additional information on reducing uncoalesced device memory accesses.
The following table lists the metrics that are key performance indicators:

Metric Name	Value	Guidance
derived__memory_l2_theoretical_sectors_global_excessive	6.69859e+06	Reduce the number of excessive wavefronts in L2