	Result	Size	Time	Cycles	GPU	SM Frequency	Process	Attributes
Current	600 - draw_lines_kernel	(1, 1, 1)x(16, 16, 1)	170.02 us	247,049	0 - NVIDIA GeForce GTX 1660 Ti	1.45 Ghz	[11724] image	

Summary | Details | Source | Context | Comments | Raw | Session

Compare | Tools | View | Export

This table shows all results in the report. Use the column headers to sort the results in this report. Double-click a result to see detailed metrics. Double-click on demangled names to rename it.

ID	Estimated Speedup [%]	Function Name	Demangled Name	Duration [ms] (6.45 ms)	Runtime Improvement [ms] (4.27 ms)	Compute Throughput [%]	Memory Throughput [%]	# Reg
6	64.29	hough_kernel	hough_kernel..char..	0.89	0.57	83.52	17.84	
7	60.69	getlines_kernel	getlines_kernel..un...	0.02	0.01	17.68	60.94	
8	95.83	draw_lines_kernel	draw_lines_kernel..	0.17	0.16	0.69	0.62	

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.*

Small Grid
Est. Speedup: 95.83%

The grid for this launch is configured to execute only 1 block, which is less than the GPU's 24 multiprocessors. This can underutilize some multiprocessors. If you do not intend to execute this kernel concurrently with other workloads, consider reducing the block size to have at least one block per multiprocessor or increase the size of the grid to fully utilize the available hardware resources. See the Hardware Model description for more details on launch configurations.

Achieved Occupancy
Est. Speedup: 82.40%

The difference between calculated theoretical (100.0%) and measured achieved occupancy (9.4%) 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.

Wait Stalls
Est. Speedup: 44.55%

On average, each warp of this workload spends 2.5 cycles being stalled waiting on a fixed latency execution dependency. Typically, this stall reason should be very low and only shows up as a top contributor in already highly optimized kernels. Try to hide the corresponding instruction latencies by increasing the number of active warps, restructuring the code or unrolling loops. Furthermore, consider switching to lower-latency instructions, e.g. by making use of fast math compiler options. This stall type represents about 44.5% of the total average of 5.7 cycles between issuing two instructions.