Analysis Report

dtopo_vel_112(float*, float*, float*, float const *, float, float, int, int, int, int, int, int, int)

Duration	20.05992 ms (20,059,915 ns)
Grid Size	[400,400,1]
Block Size	[1,1,64]
Registers/Thread	80
Shared Memory/Block	0 B
Shared Memory Executed	0 B
Shared Memory Bank Size	4 B

[0] Tesla V100-SXM2-16GB

GPU UUID	GPU-fb28a151-65e0-2cc7-3783-f424c93e2314
Compute Capability	7.0
Max. Threads per Block	1024
Max. Threads per Multiprocessor	2048
Max. Shared Memory per Block	48 KiB
Max. Shared Memory per Multiprocessor	96 KiB
Max. Registers per Block	65536
Max. Registers per Multiprocessor	65536
Max. Grid Dimensions	[2147483647, 65535, 65535]
Max. Block Dimensions	[1024, 1024, 64]
Max. Warps per Multiprocessor	64
Max. Blocks per Multiprocessor	32
Half Precision FLOP/s	31.334 TeraFLOP/s
Single Precision FLOP/s	15.667 TeraFLOP/s
Double Precision FLOP/s	7.834 TeraFLOP/s
Number of Multiprocessors	80
Multiprocessor Clock Rate	1.53 GHz
Concurrent Kernel	true
Max IPC	4
Threads per Warp	32
Global Memory Bandwidth	898.048 GB/s
Global Memory Size	15.75 GiB
Constant Memory Size	64 KiB
L2 Cache Size	6 MiB
Memcpy Engines	4
PCIe Generation	3

[0] Tesla V100-SXM2-16GB

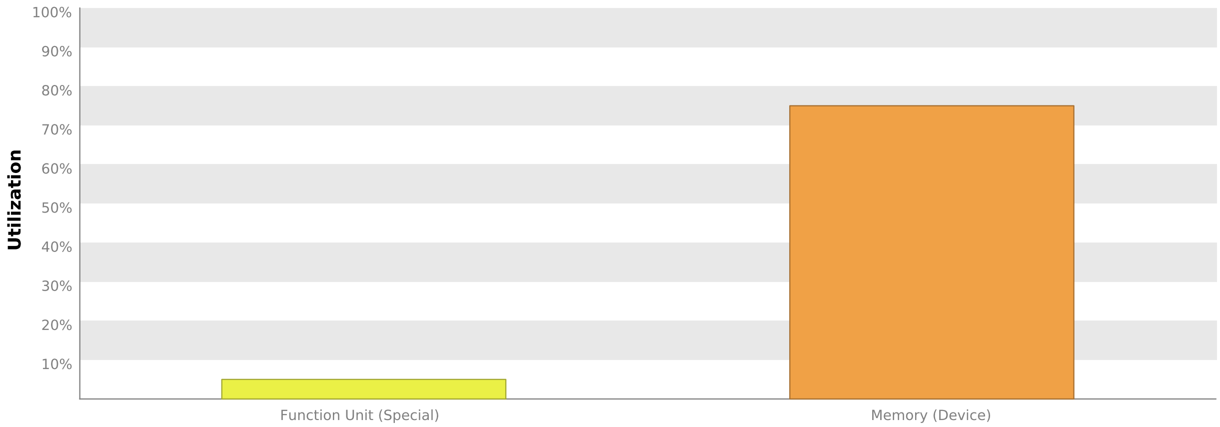
PCIe Link Rate	8 Gbit/s
PCIe Link Width	8

1. Compute, Bandwidth, or Latency Bound

The first step in analyzing an individual kernel is to determine if the performance of the kernel is bounded by computation, memory bandwidth, or instruction/memory latency. The results below indicate that the performance of kernel "dtopo_vel_112" is most likely limited by memory bandwidth. You should first examine the information in the "Memory Bandwidth" section to determine how it is limiting performance.

1.1. Kernel Performance Is Bound By Memory Bandwidth

For device "Tesla V100-SXM2-16GB" the kernel's compute utilization is significantly lower than its memory utilization. These utilization levels indicate that the performance of the kernel is most likely being limited by the memory system. For this kernel the limiting factor in the memory system is the bandwidth of the Device memory.



2. Memory Bandwidth

Memory bandwidth limits the performance of a kernel when one or more memories in the GPU cannot provide data at the rate requested by the kernel. The results below indicate that the kernel is limited by the bandwidth available to the device memory.

2.1. High Local Memory Overhead

Local memory loads and stores account for 96% of total memory traffic. High local memory traffic typically indicates excessive register spilling.

Optimization: Use the -maxrregcount flag or the __launch_bounds__ qualifier to increase the number of registers available to nvcc when compiling the kernel.

2.2. GPU Utilization Is Limited By Memory Bandwidth

The following table shows the memory bandwidth used by this kernel for the various types of memory on the device. The table also shows the utilization of each memory type relative to the maximum throughput supported by the memory. The results show that the kernel's performance is potentially limited by the bandwidth available from one or more of the memories on the device.

Optimization: Try the following optimizations for the memory with high bandwidth utilization.
Shared Memory - If possible use 64-bit accesses to shared memory and 8-byte bank mode to achieved 2x throughput.
L2 Cache - Align and block kernel data to maximize L2 cache efficiency.
Unified Cache - Reallocate texture data to shared or global memory. Resolve alignment and access pattern issues for global loads and stores.
Device Memory - Resolve alignment and access pattern issues for global loads and stores.
System Memory (via PCIe) - Make sure performance critical data is placed in device or shared memory.

Transactions	Bandwidth	Utilization	
Shared Memory			
Shared Loads	0	0 B/s	
Shared Stores	0	0 B/s	
Shared Total	0	0 B/s	Idle Low Medium High Max
L2 Cache			
Reads	52003094	82.956 GB/s	
Writes	315417992	503.161 GB/s	
Total	367421086	586.118 GB/s	Idle Low Medium High Max
Unified Cache			
Local Loads	37785600	60.276 GB/s	
Local Stores	314880000	502.303 GB/s	
Global Loads	62054400	98.99 GB/s	
Global Stores	460800	735.078 MB/s	
Texture Reads	79079597	504.598 GB/s	
Unified Total	494260397	1,166.903 GB/s	Idle Low Medium High Max
Device Memory			
Reads	91929823	146.648 GB/s	
Writes	314753761	502.102 GB/s	
Total	406683584	648.75 GB/s	Idle Low Medium High Max
System Memory			
[PCIe configuration: Gen3 x8, 8 Gbit/s]			
Reads	0	0 B/s	Idle Low Medium High Max
Writes	5	7.976 kB/s	Idle Low Medium High Max

2.3. Memory Statistics

The following chart shows a summary view of the memory hierarchy of the CUDA programming model. The green nodes in the diagram depict logical memory space whereas blue nodes depicts actual hardware unit on the chip. For the various caches the reported percentage number states the cache hit rate; that is the ratio of requests that could be served with data locally available to the cache over all requests made.

The links between the nodes in the diagram depict the data paths between the SMs to the memory spaces into the memory system. Different metrics are shown per data path. The data paths from the SMs to the memory spaces report the total number of memory instructions executed, it includes both read and write operations. The data path between memory spaces and "Unified Cache" or "Shared Memory" reports the total amount of memory requests made (read or write). All other data paths report the total amount of transferred memory in bytes.

3. Instruction and Memory Latency

Instruction and memory latency limit the performance of a kernel when the GPU does not have enough work to keep busy. The performance of latency-limited kernels can often be improved by increasing occupancy. Occupancy is a measure of how many warps the kernel has active on the GPU, relative to the maximum number of warps supported by the GPU. Theoretical occupancy provides an upper bound while achieved occupancy indicates the kernel's actual occupancy. The results below indicate that occupancy can be improved by reducing the number of registers used by the kernel.

3.1. GPU Utilization Is Limited By Register Usage

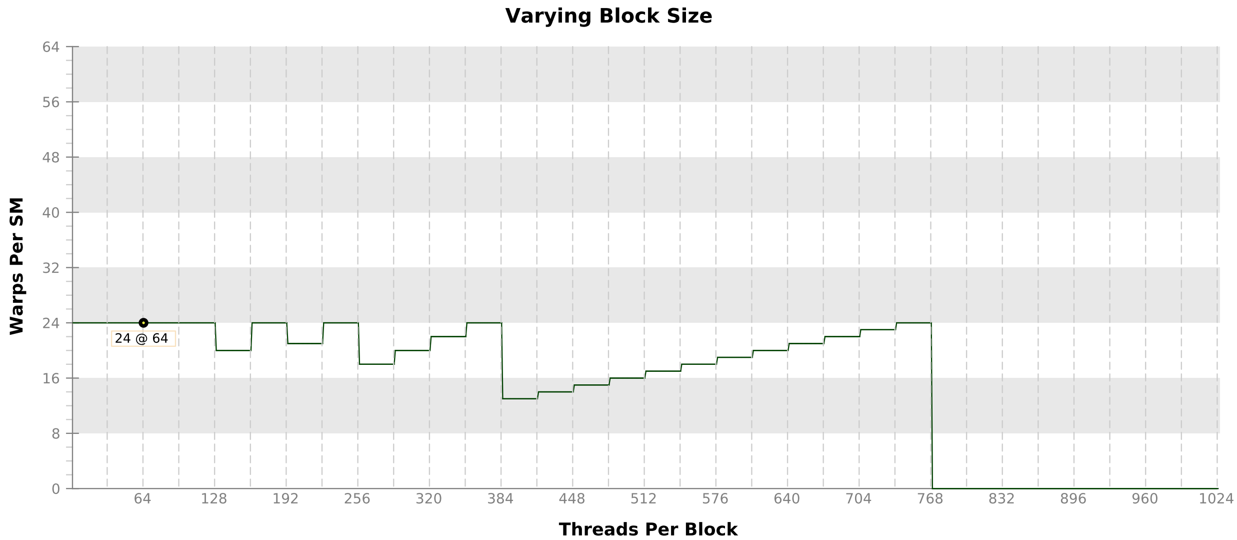
The kernel uses 80 registers for each thread (5120 registers for each block). This register usage is likely preventing the kernel from fully utilizing the GPU. Device "Tesla V100-SXM2-16GB" provides up to 65536 registers for each block. Because the kernel uses 5120 registers for each block each SM is limited to simultaneously executing 12 blocks (24 warps). Chart "Varying Register Count" below shows how changing register usage will change the number of blocks that can execute on each SM.

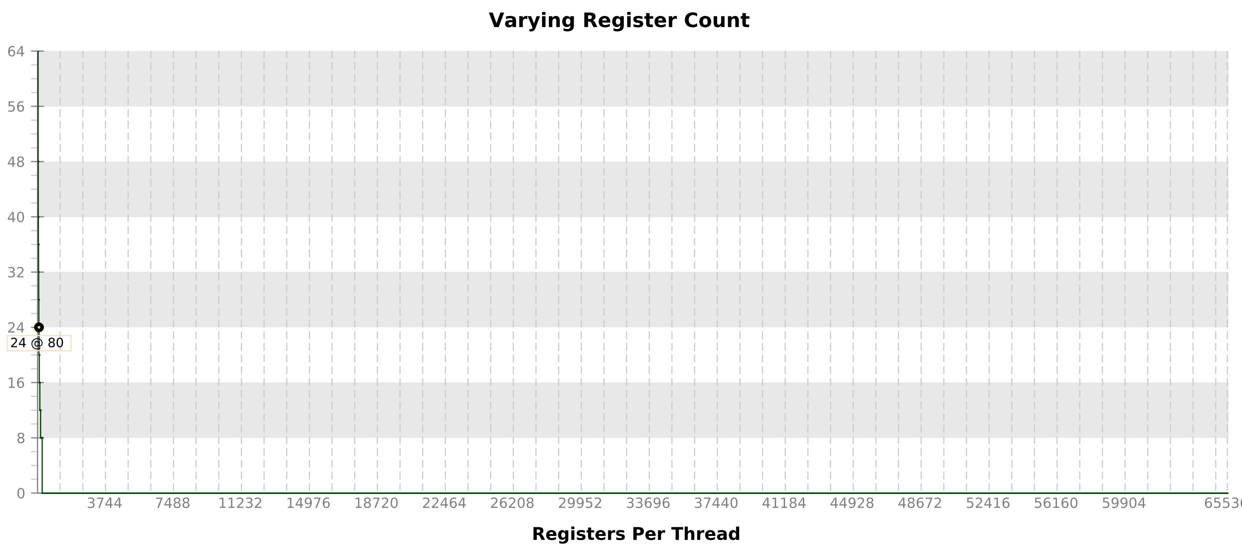
Optimization: Use the -maxrregcount flag or the __launch_bounds__ qualifier to decrease the number of registers used by each thread. This will increase the number of blocks that can execute on each SM. On devices with Compute Capability 5.2 turning global cache off can increase the occupancy limited by register usage.

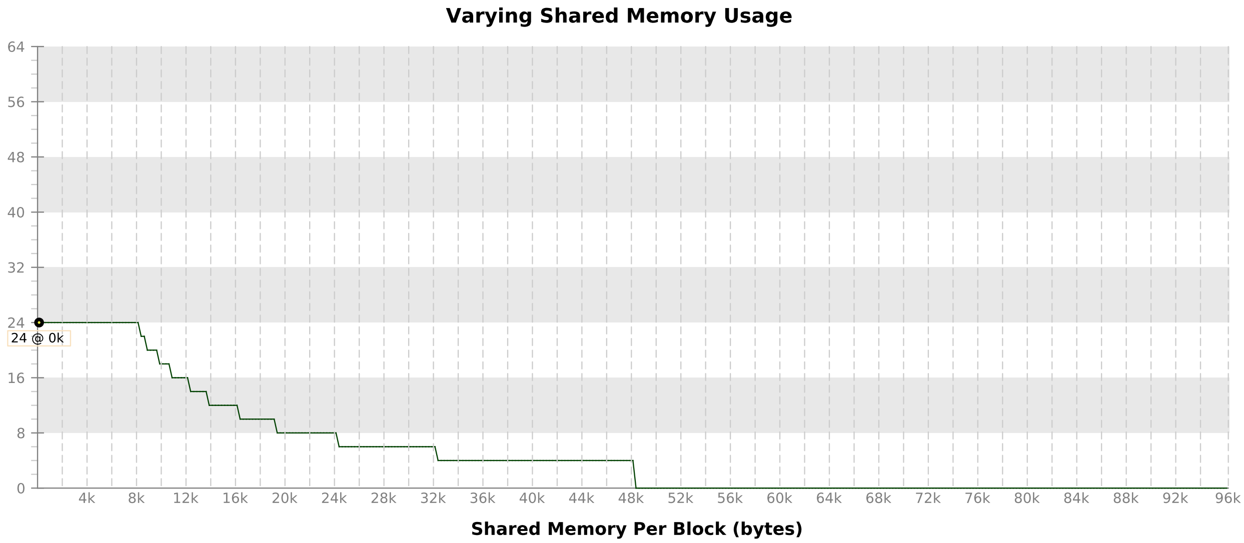
Variable	Achieved	Theoretical	Device Limit	Grid Size: [400,400,1] (160000 blocks) Block Size: [1,1
Occupancy Per SM				
Active Blocks		12	32	0 4 8 12 16 20 24 28 32
Active Warps	22.22	24	64	0 9 18 27 36 45 54 6364
Active Threads		768	2048	0 512 1024 1536 2048
Occupancy	34.7%	37.5%	100%	0% 25% 50% 75% 100%
Warps				
Threads/Block		64	1024	0 256 512 768 1024
Warps/Block		2	32	0 4 8 12 16 20 24 28 32
Block Limit		32	32	0 4 8 12 16 20 24 28 32
Registers				
Registers/Thread		80	65536	0 16384 32768 49152 65536
Registers/Block		5120	65536	0 16k 32k 48k 64k
Block Limit		12	32	0 4 8 12 16 20 24 28 32
Shared Memory				
Shared Memory/Block		0	98304	0 32k 64k 96k
Block Limit		0	32	0 4 8 12 16 20 24 28 32

3.2. Occupancy Charts

The following charts show how varying different components of the kernel will impact theoretical occupancy.

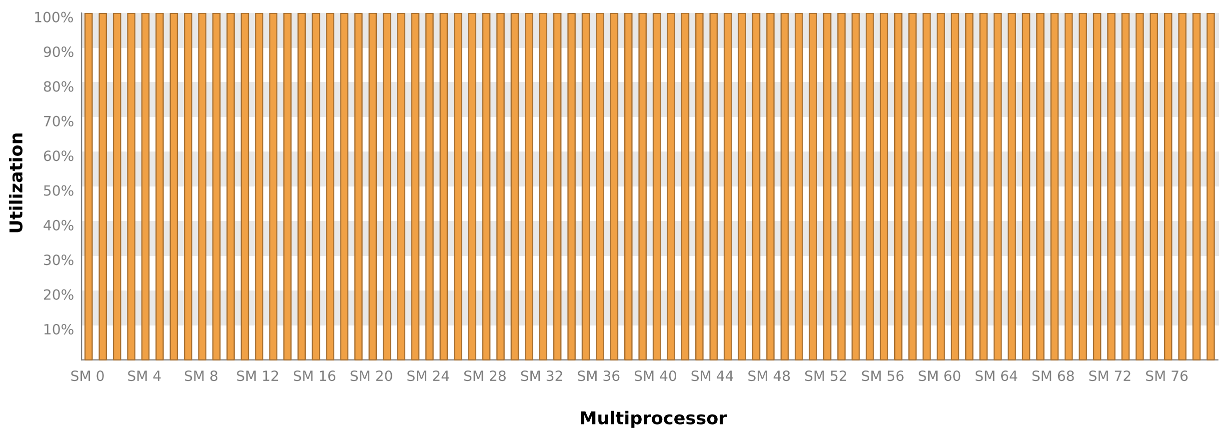

Varying Block Size
Warps Per SM
Threads Per Block
24 @ 64
0
8
16
24
32
40
48
56
64
64
128
192
256
320
384
448
512
576
640
704
768
832
896
960
1024


Varying Register Count
Registers Per Thread
24 @ 80
0
8
16
24
32
40
48
56
64
3744
7488
11232
14976
18720
22464
26208
29952
33696
37440
41184
44928
48672
52416
56160
59904
65536



3.3. Multiprocessor Utilization

The kernel's blocks are distributed across the GPU's multiprocessors for execution. Depending on the number of blocks and the execution duration of each block some multiprocessors may be more highly utilized than others during execution of the kernel. The following chart shows the utilization of each multiprocessor during execution of the kernel.



4. Compute Resources

GPU compute resources limit the performance of a kernel when those resources are insufficient or poorly utilized. Compute resources are used most efficiently when all threads in a warp have the same branching and predication behavior. The results below indicate that a significant fraction of the available compute performance is being wasted because branch and predication behavior is differing for threads within a warp.

4.1. Low Warp Execution Efficiency

Warp execution efficiency is the average percentage of active threads in each executed warp. Increasing warp execution efficiency will increase utilization of the GPU's compute resources. The warp execution efficiency for these kernels is 44.4% if predicated instructions are not taken into account. The kernel's not predicated off warp execution efficiency of 44% is less than 100% due to divergent branches and predicated instructions.

Optimization: Reduce the amount of intra-warp divergence and predication in the kernel.

4.2. Divergent Branches

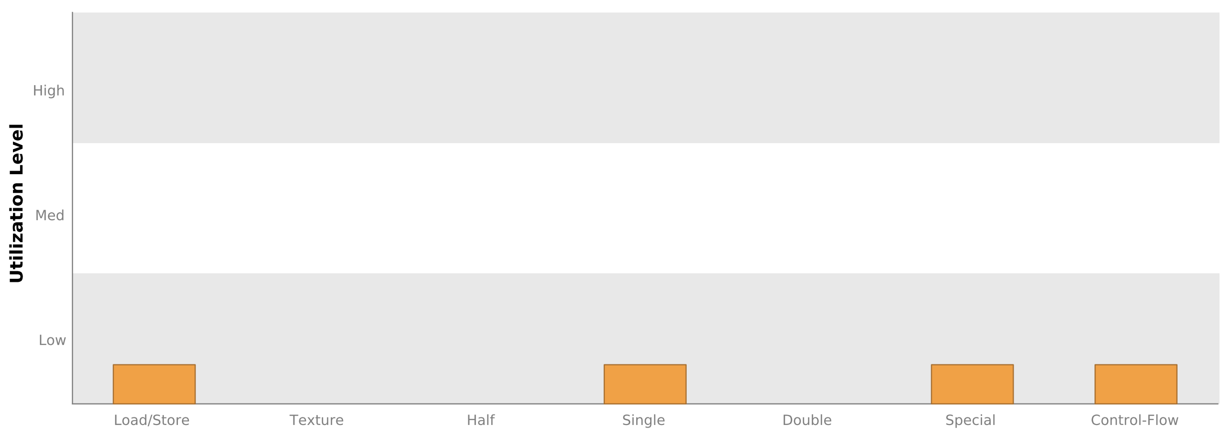
Compute resource are used most efficiently when all threads in a warp have the same branching behavior. When this does not occur the branch is said to be divergent. Divergent branches lower warp execution efficiency which leads to inefficient use of the GPU's compute resources.

Optimization: Each entry below points to a divergent branch within the kernel. For each branch reduce the amount of intra-warp divergence.

4.3. Function Unit Utilization

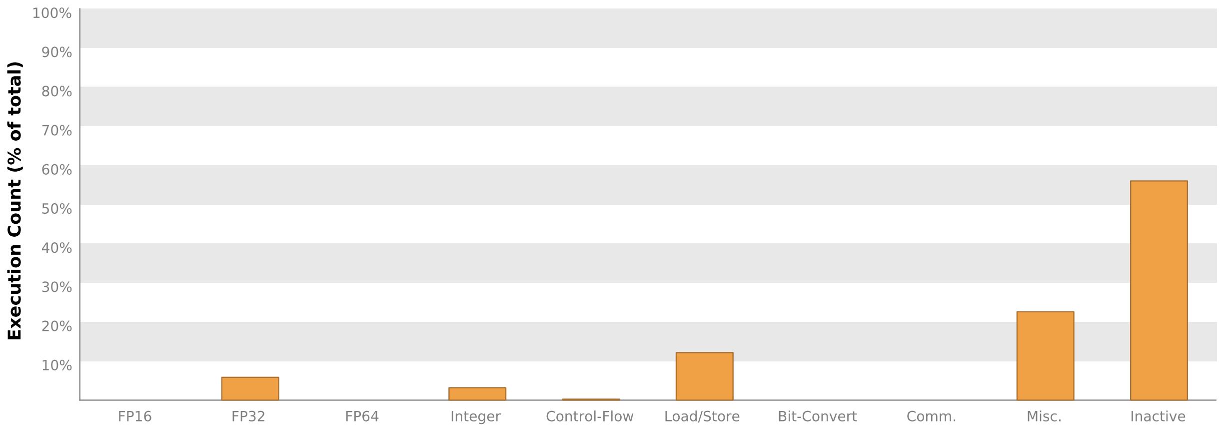
Different types of instructions are executed on different function units within each SM. Performance can be limited if a function unit is over-used by the instructions executed by the kernel. The following results show that the kernel's performance is not limited by overuse of any function unit.

Load/Store - Load and store instructions for shared and constant memory.
Texture - Load and store instructions for local, global, and texture memory.
Half - Half-precision floating-point arithmetic instructions.
Single - Single-precision integer and floating-point arithmetic instructions.
Double - Double-precision floating-point arithmetic instructions.
Special - Special arithmetic instructions such as sin, cos, popc, etc.
Control-Flow - Direct and indirect branches, jumps, and calls.



4.4. Instruction Execution Counts

The following chart shows the mix of instructions executed by the kernel. The instructions are grouped into classes and for each class the chart shows the percentage of thread execution cycles that were devoted to executing instructions in that class. The "Inactive" result shows the thread executions that did not execute any instruction because the thread was predicated or inactive due to divergence.



4.5. Floating-Point Operation Counts

The following chart shows the mix of floating-point operations executed by the kernel. The operations are grouped into classes and for each class the chart shows the percentage of thread execution cycles that were devoted to executing operations in that class. The results do not sum to 100% because non-floating-point operations executed by the kernel are not shown in this chart.