Analysis Report

mxnet::op::forward_kernel(float*, float const *, float const *, int, int, int, int, int, int)

Duration	82.16049 ms (82,160,486 ns)
Grid Size	[10000,24,1]
Block Size	[32,32,1]
Registers/Thread	32
Shared Memory/Block	0 B
Shared Memory Executed	0 B
Shared Memory Bank Size	4 B

[0] TITAN V

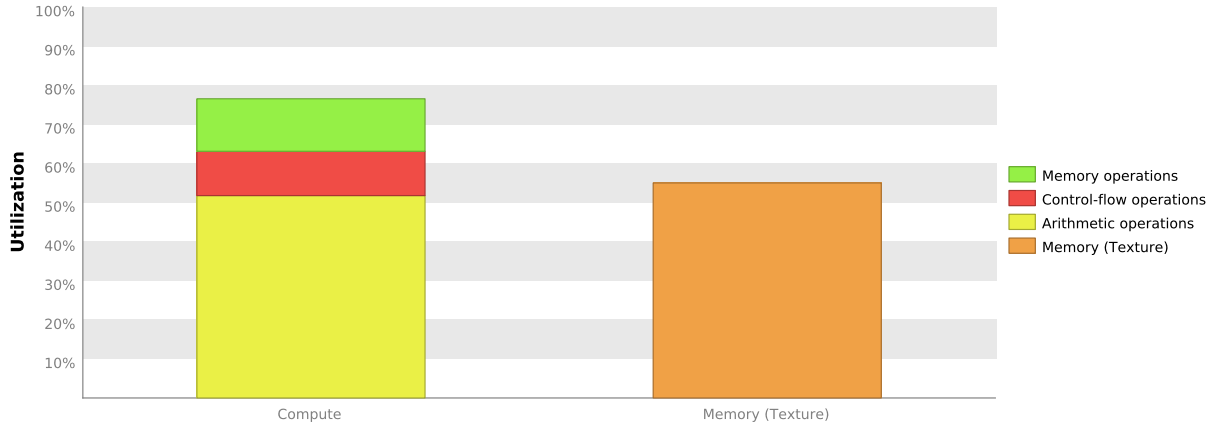
GPU UUID	GPU-581188a1-e085-765c-7577-471062aebf7e
Compute Capability	7.0
Max. Threads per Block	1024
Max. Threads per Multiprocessor	2048
Max. Shared Memory per Block	48 KiB
Max. Shared Memory per Multiprocessor	96 KiB
Max. Registers per Block	65536
Max. Registers per Multiprocessor	65536
Max. Grid Dimensions	[2147483647, 65535, 65535]
Max. Block Dimensions	[1024, 1024, 64]
Max. Warps per Multiprocessor	64
Max. Blocks per Multiprocessor	32
Half Precision FLOP/s	29.798 TeraFLOP/s
Single Precision FLOP/s	14.899 TeraFLOP/s
Double Precision FLOP/s	7.45 TeraFLOP/s
Number of Multiprocessors	80
Multiprocessor Clock Rate	1.455 GHz
Concurrent Kernel	true
Max IPC	4
Threads per Warp	32
Global Memory Bandwidth	652.8 GB/s
Global Memory Size	11.752 GiB
Constant Memory Size	64 KiB
L2 Cache Size	4.5 MiB
Memcpy Engines	7
PCIe Generation	3
PCIe Link Rate	8 Gbit/s
PCIe Link Width	16

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 "mxnet::op::forward_kernel" is most likely limited by both compute and memory bandwidth. You should first examine the information in the "Compute Resources" section to determine how it is limiting performance.

1.1. Kernel Performance Is Bound By Compute And Memory Bandwidth

For device "TITAN V" compute and memory utilization are balanced. These utilization levels indicate that kernel performance is good, but that additional performance improvement may be possible if either of both of compute and memory utilization levels are increased.



2. 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.

2.1. 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.

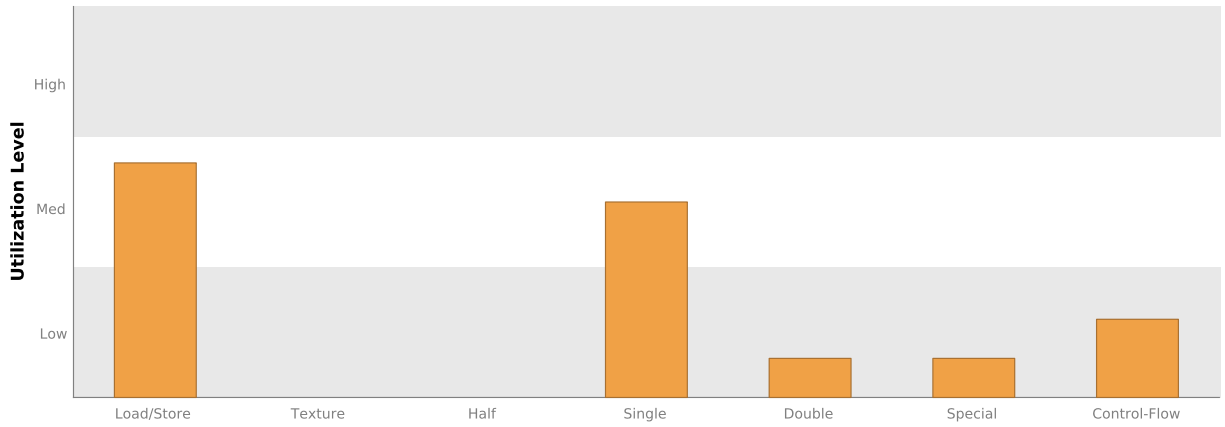
/mxnet/src/operator/custom/./new-forward.cuh

Line 39	Divergence = 90.6% [6960000 divergent executions out of 7680000 total executions]
Line 42	Divergence = 0% [0 divergent executions out of 83520000 total executions]
Line 42	Divergence = 0% [0 divergent executions out of 6960000 total executions]
Line 43	Divergence = 0% [0 divergent executions out of 83520000 total executions]
Line 43	Divergence = 0% [0 divergent executions out of 417600000 total executions]
Line 44	Divergence = 0% [0 divergent executions out of 417600000 total executions]
Line 44	Divergence = 0% [0 divergent executions out of 417600000 total executions]
Line 44	Divergence = 0% [0 divergent executions out of 417600000 total executions]
Line 44	Divergence = 0% [0 divergent executions out of 417600000 total executions]
Line 44	Divergence = 0% [0 divergent executions out of 417600000 total executions]
Line 44	Divergence = 0% [0 divergent executions out of 417600000 total executions]
Line 45	Divergence = 0% [0 divergent executions out of 417600000 total executions]
Line 45	Divergence = 0% [0 divergent executions out of 417600000 total executions]
Line 54	Divergence = 0% [0 divergent executions out of 6960000 total executions]

2.2. Function Unit Utilization

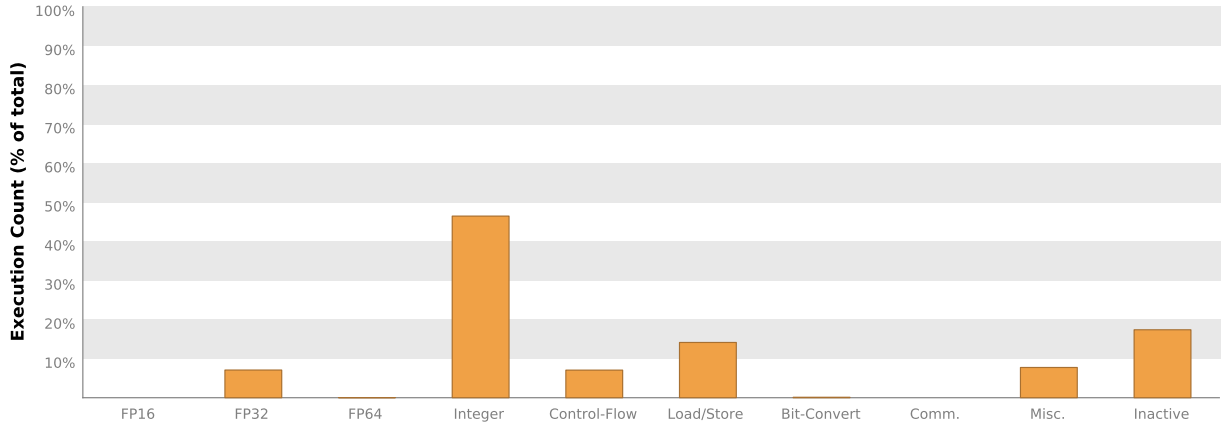
Different types of instructions are executed on different function units within each SM. Performance can be limited if a function unit is over-used by the instructions executed by the kernel. The following results show that the kernel's performance is not limited by overuse of any function unit.

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.



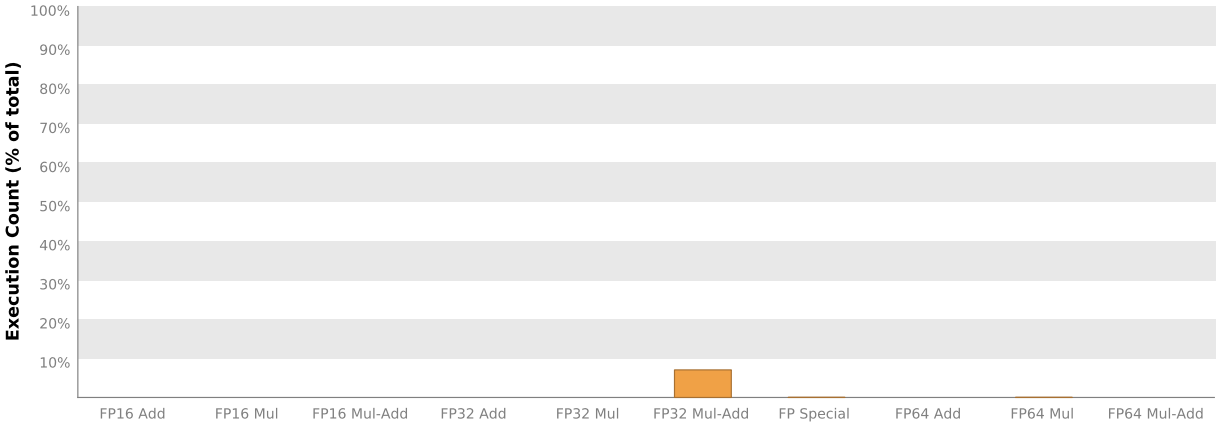
2.3. Instruction Execution Counts

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.



2.4. 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.


Execution Count (% of total)
100%
90%
80%
70%
60%
50%
40%
30%
20%
10%
FP16 Add
FP16 Mul
FP16 Mul-Add
FP32 Add
FP32 Mul
FP32 Mul-Add
FP Special
FP64 Add
FP64 Mul
FP64 Mul-Add

3. 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 unified cache that holds texture, global, and local data.

3.1. Global Memory Alignment and Access Pattern

Memory bandwidth is used most efficiently when each global memory load and store has proper alignment and access pattern. The analysis is per assembly instruction.

Optimization: Each entry below points to a global load or store within the kernel with an inefficient alignment or access pattern. For each load or store improve the alignment and access pattern of the memory access.

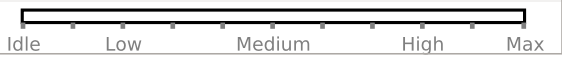
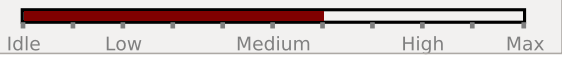
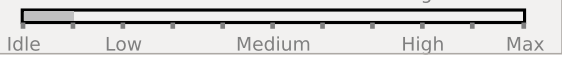
/mxnet/src/operator/custom/./new-forward.cuh

Line 45	Global Load L2 Transactions/Access = 1, Ideal Transactions/Access = 1 [417600000 L2 transactions for 417600000 total executions]
Line 45	Global Load L2 Transactions/Access = 1, Ideal Transactions/Access = 1 [417600000 L2 transactions for 417600000 total executions]
Line 45	Global Load L2 Transactions/Access = 4.5, Ideal Transactions/Access = 4 [1879200000 L2 transactions for 417600000 total executions]
Line 45	Global Load L2 Transactions/Access = 4.5, Ideal Transactions/Access = 4 [1879200000 L2 transactions for 417600000 total executions]
Line 45	Global Load L2 Transactions/Access = 4.5, Ideal Transactions/Access = 4 [1879200000 L2 transactions for 417600000 total executions]
Line 45	Global Load L2 Transactions/Access = 1, Ideal Transactions/Access = 1 [417600000 L2 transactions for 417600000 total executions]
Line 45	Global Load L2 Transactions/Access = 1, Ideal Transactions/Access = 1 [417600000 L2 transactions for 417600000 total executions]
Line 45	Global Load L2 Transactions/Access = 4.5, Ideal Transactions/Access = 4 [1879200000 L2 transactions for 417600000 total executions]
Line 45	Global Load L2 Transactions/Access = 4.5, Ideal Transactions/Access = 4 [1879200000 L2 transactions for 417600000 total executions]
Line 45	Global Load L2 Transactions/Access = 1, Ideal Transactions/Access = 1 [417600000 L2 transactions for 417600000 total executions]
Line 49	Global Store L2 Transactions/Access = 4.5, Ideal Transactions/Access = 4 [31320000 L2 transactions for 6960000 total executions]

3.2. 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	
L2 Cache			
Reads	392588297	152.906 GB/s	
Writes	31320510	12.199 GB/s	
Total	423908807	165.105 GB/s	
Unified Cache			
Local Loads	0	0 B/s	
Local Stores	0	0 B/s	
Global Loads	11439923270	4,455.64 GB/s	
Global Stores	31320000	12.199 GB/s	
Texture Reads	4490496542	6,995.864 GB/s	
Unified Total	15961739812	11,463.702 GB/s	
Device Memory			
Reads	392424480	152.842 GB/s	
Writes	25489856	9.928 GB/s	
Total	417914336	162.77 GB/s	
System Memory			
[PCIe configuration: Gen3 x16, 8 Gbit/s]			
Reads	0	0 B/s	
Writes	5	1.947 kB/s	

3.3. Memory Statistics

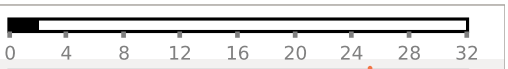
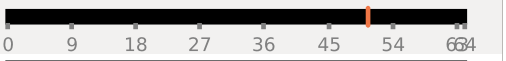
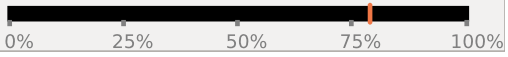
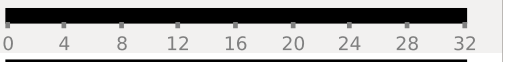
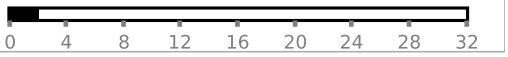
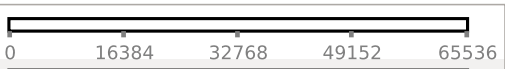
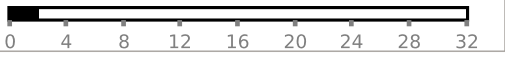
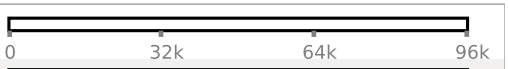
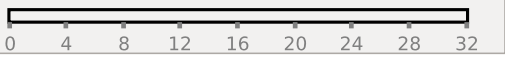
The following chart shows a summary view of the memory hierarchy of the CUDA programming model. The green nodes in the diagram depict logical memory space whereas blue nodes depicts actual hardware unit on the chip. For the various caches the reported percentage number states the cache hit rate; that 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.

4. 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.

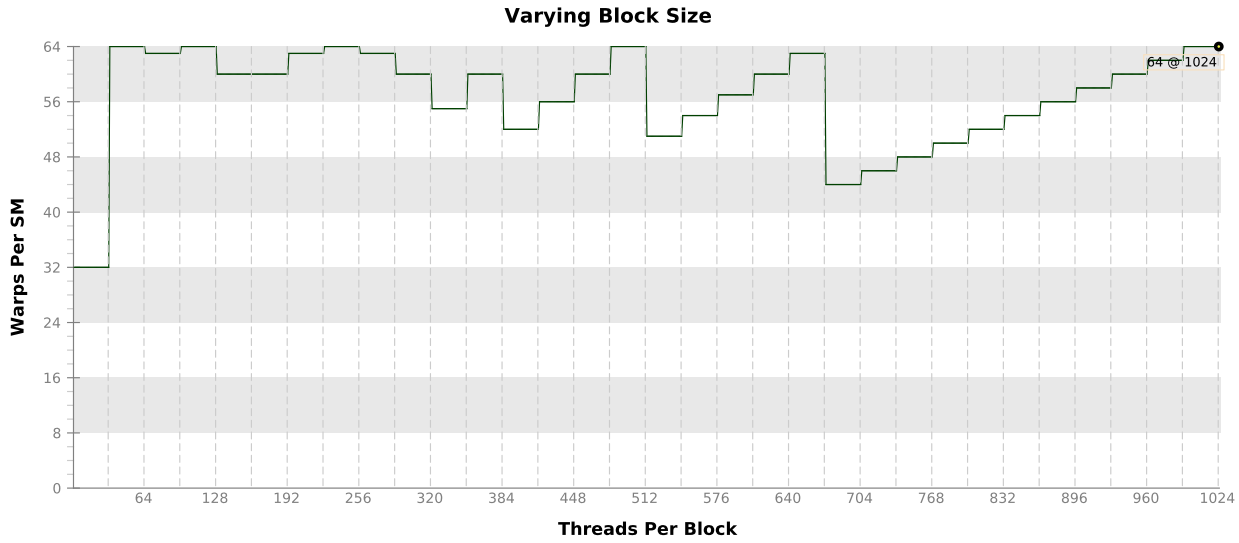
4.1. Occupancy Is Not Limiting Kernel Performance

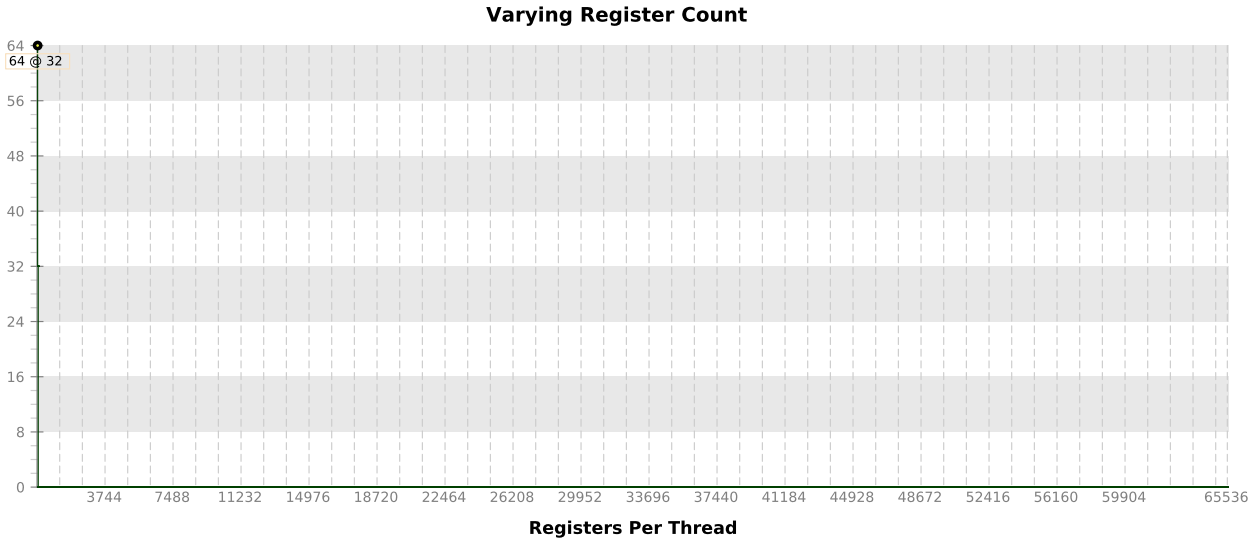
The kernel's block size, register usage, and shared memory usage allow it to fully utilize all warps on the GPU.

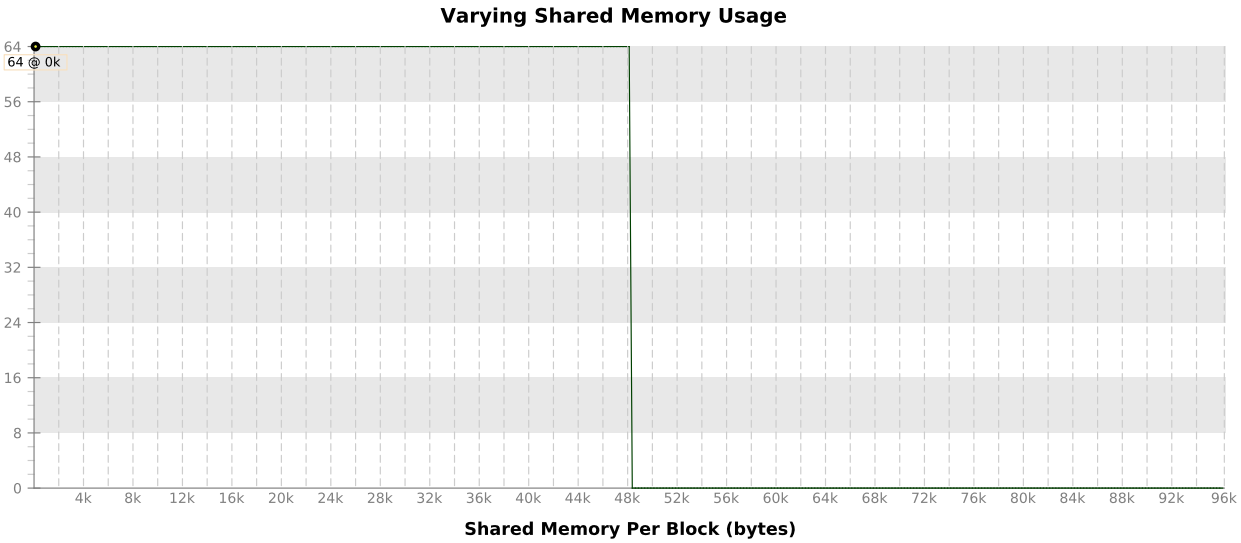
Variable	Achieved	Theoretical	Device Limit	Grid Size: [10000,24,1] (240000 blocks) Block Size: [3
Occupancy Per SM				
Active Blocks		2	32	
Active Warps	50.05	64	64	
Active Threads		2048	2048	
Occupancy	78.2%	100%	100%	
Warps				
Threads/Block		1024	1024	
Warps/Block		32	32	
Block Limit		2	32	
Registers				
Registers/Thread		32	65536	
Registers/Block		32768	65536	
Block Limit		2	32	
Shared Memory				
Shared Memory/Block		0	98304	
Block Limit		0	32	

4.2. Occupancy Charts

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


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


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



4.3. 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.