

# Analysis Report

**mxnet::op::forward\_kernel\_shared(float\*, float const \*, float const \*, int, int, int, int, int, int)**

Duration	12.35839 ms (12,358,385 ns)
Grid Size	[ 1000,24,4 ]
Block Size	[ 16,16,1 ]
Registers/Thread	40
Shared Memory/Block	1.66 KiB
Shared Memory Executed	0 B
Shared Memory Bank Size	4 B

## [0] TITAN V

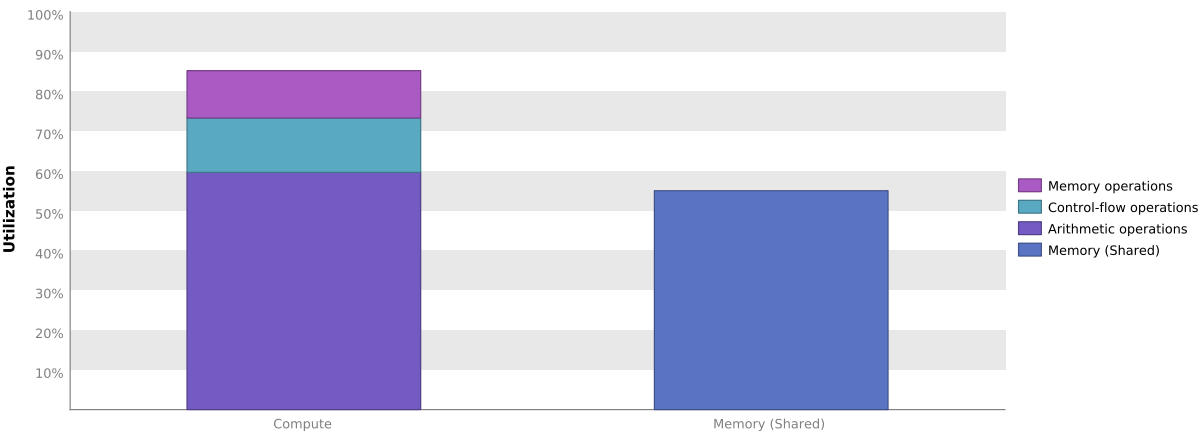
GPU UUID	GPU-9910c808-3335-47c7-980a-bf909aedc5aa
Compute Capability	7.0
Max. Threads per Block	1024
Max. Threads per Multiprocessor	2048
Max. Shared Memory per Block	48 KiB
Max. Shared Memory per Multiprocessor	96 KiB
Max. Registers per Block	65536
Max. Registers per Multiprocessor	65536
Max. Grid Dimensions	[ 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	11.755 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	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 "mxnet::op::forward\_kernel\_s..." is most likely limited by compute. 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

For device "TITAN V" the kernel's memory utilization is significantly lower than its compute utilization. These utilization levels indicate that the performance of the kernel is most likely being limited by computation on the SMs.



## 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 42	Divergence = 0% [ 0 divergent executions out of 9216000 total executions ]
Line 42	Divergence = 0% [ 0 divergent executions out of 768000 total executions ]
Line 43	Divergence = 37.5% [ 3456000 divergent executions out of 9216000 total executions ]
Line 47	Divergence = 0% [ 0 divergent executions out of 9216000 total executions ]
Line 47	Divergence = 50% [ 9216000 divergent executions out of 18432000 total executions ]
Line 48	Divergence = 0% [ 0 divergent executions out of 18432000 total executions ]
Line 48	Divergence = 0% [ 0 divergent executions out of 13824000 total executions ]
Line 48	Divergence = 0% [ 0 divergent executions out of 18432000 total executions ]
Line 48	Divergence = 0% [ 0 divergent executions out of 18432000 total executions ]
Line 48	Divergence = 0% [ 0 divergent executions out of 18432000 total executions ]
Line 53	Divergence = 0% [ 0 divergent executions out of 9216000 total executions ]
Line 53	Divergence = 0% [ 0 divergent executions out of 46080000 total executions ]
Line 54	Divergence = 0% [ 0 divergent executions out of 46080000 total executions ]
Line 54	Divergence = 0% [ 0 divergent executions out of 46080000 total executions ]
Line 54	Divergence = 0% [ 0 divergent executions out of 46080000 total executions ]
Line 54	Divergence = 0% [ 0 divergent executions out of 46080000 total executions ]
Line 54	Divergence = 0% [ 0 divergent executions out of 46080000 total executions ]
Line 54	Divergence = 0% [ 0 divergent executions out of 46080000 total executions ]
Line 55	Divergence = 0% [ 0 divergent executions out of 46080000 total executions ]
Line 55	Divergence = 0% [ 0 divergent executions out of 46080000 total executions ]
Line 66	Divergence = 0% [ 0 divergent executions out of 768000 total executions ]

### 2.2. Function Unit Utilization

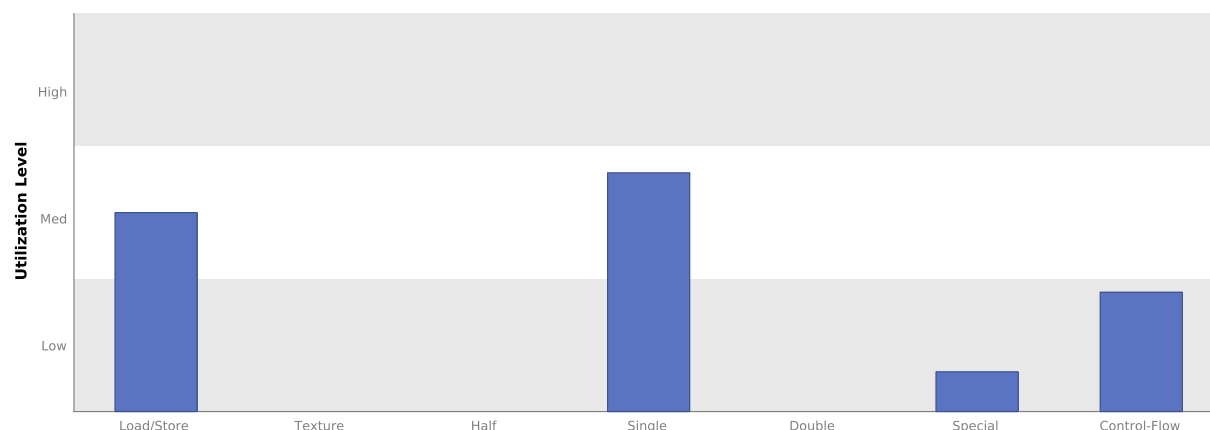
Different types of instructions are executed on different function units within each SM. Performance can be limited if a function unit is over-used by the instructions executed by the kernel. The following results show that the kernel's performance is not limited by overuse of any function unit.

Load/Store - Load and store instructions for shared and constant memory.

Texture - Load and store instructions for local, global, and texture memory.

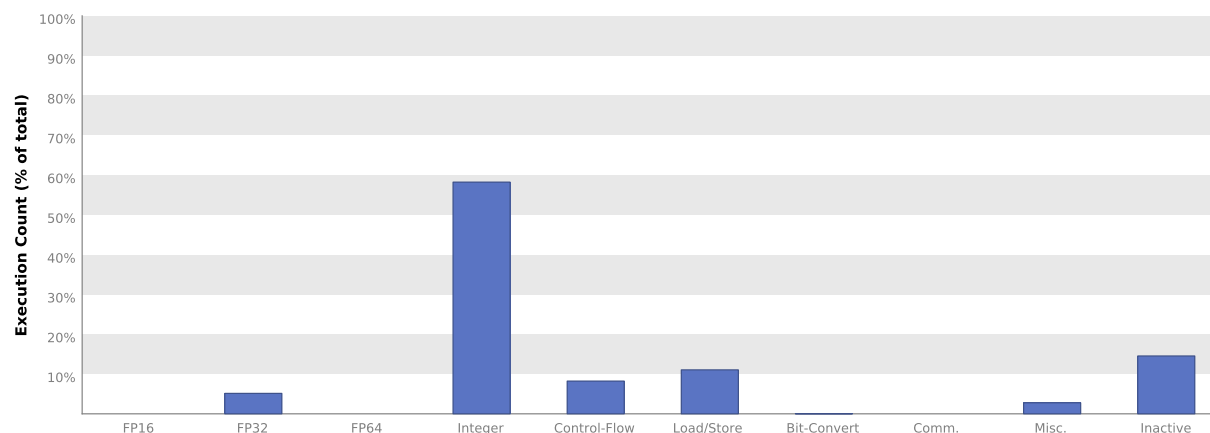
Half - Half-precision floating-point arithmetic instructions.

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.



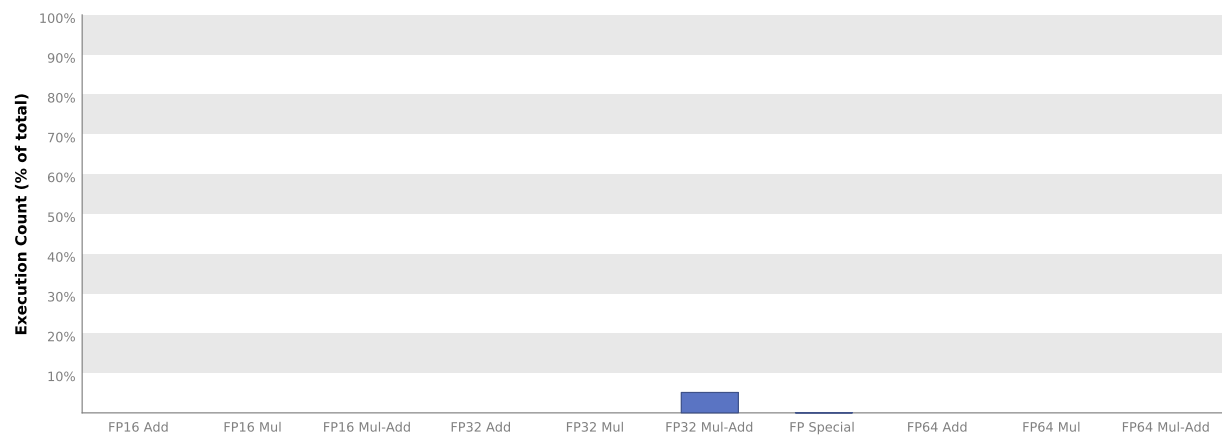
## 2.3. Instruction Execution Counts

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.



### 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 shared memory.

#### 3.1. Shared Memory Alignment and Access Pattern

Memory bandwidth is used most efficiently when each shared memory load and store has proper alignment and access pattern.



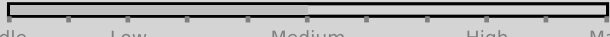


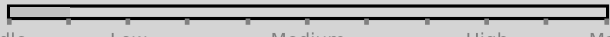
*Optimization: Select each entry below to open the source code to a shared load or store within the kernel with an inefficient alignment or access pattern. For each load or store improve the alignment and access pattern of the memory access.*

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

Line 44	Shared Store Transactions/Access = 1, Ideal Transactions/Access = 1 [ 3456000 transactions for 3456000 total executions ]
Line 49	Shared Store Transactions/Access = 1.5, Ideal Transactions/Access = 1 [ 6912000 transactions for 4608000 total executions ]
Line 49	Shared Store Transactions/Access = 1.5, Ideal Transactions/Access = 1 [ 20736000 transactions for 13824000 total executions ]
Line 49	Shared Store Transactions/Access = 1.5, Ideal Transactions/Access = 1 [ 6912000 transactions for 4608000 total executions ]
Line 55	Shared Load Transactions/Access = 1, Ideal Transactions/Access = 1 [ 46080000 transactions for 46080000 total executions ]
Line 55	Shared Load Transactions/Access = 2, Ideal Transactions/Access = 1 [ 92160000 transactions for 46080000 total executions ]
Line 55	Shared Load Transactions/Access = 2, Ideal Transactions/Access = 1 [ 92160000 transactions for 46080000 total executions ]
Line 55	Shared Load Transactions/Access = 1, Ideal Transactions/Access = 1 [ 46080000 transactions for 46080000 total executions ]
Line 55	Shared Load Transactions/Access = 1, Ideal Transactions/Access = 1 [ 46080000 transactions for 46080000 total executions ]
Line 55	Shared Load Transactions/Access = 2, Ideal Transactions/Access = 1 [ 92160000 transactions for 46080000 total executions ]
Line 55	Shared Load Transactions/Access = 2, Ideal Transactions/Access = 1 [ 92160000 transactions for 46080000 total executions ]
Line 55	Shared Load Transactions/Access = 1, Ideal Transactions/Access = 1 [ 46080000 transactions for 46080000 total executions ]
Line 55	Shared Load Transactions/Access = 2, Ideal Transactions/Access = 1 [ 92160000 transactions for 46080000 total executions ]
Line 55	Shared Load Transactions/Access = 1, Ideal Transactions/Access = 1 [ 46080000 transactions for 46080000 total executions ]

#### 3.2. Memory Bandwidth And Utilization

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.

Transactions	Bandwidth	Utilization	
Shared Memory			
Shared Loads	714592321	7,401.276 GB/s	
Shared Stores	38727648	401.115 GB/s	
Shared Total	753319969	7,802.391 GB/s	
L2 Cache			
Reads	68308376	176.873 GB/s	
Writes	139896016	362.238 GB/s	
Total	208204392	539.111 GB/s	
Unified Cache			
Local Loads	0	0 B/s	
Local Stores	0	0 B/s	
Global Loads	238303906	617.049 GB/s	
Global Stores	139896000	362.238 GB/s	
Texture Reads	533253004	5,523.083 GB/s	
Unified Total	911452910	6,502.369 GB/s	
Device Memory			
Reads	86731986	224.578 GB/s	
Writes	4197195	10.868 GB/s	
Total	90929181	235.446 GB/s	
System Memory			
[ PCIe configuration: Gen3 x8, 8 Gbit/s ]			
Reads	0	0 B/s	
Writes	5	12.946 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

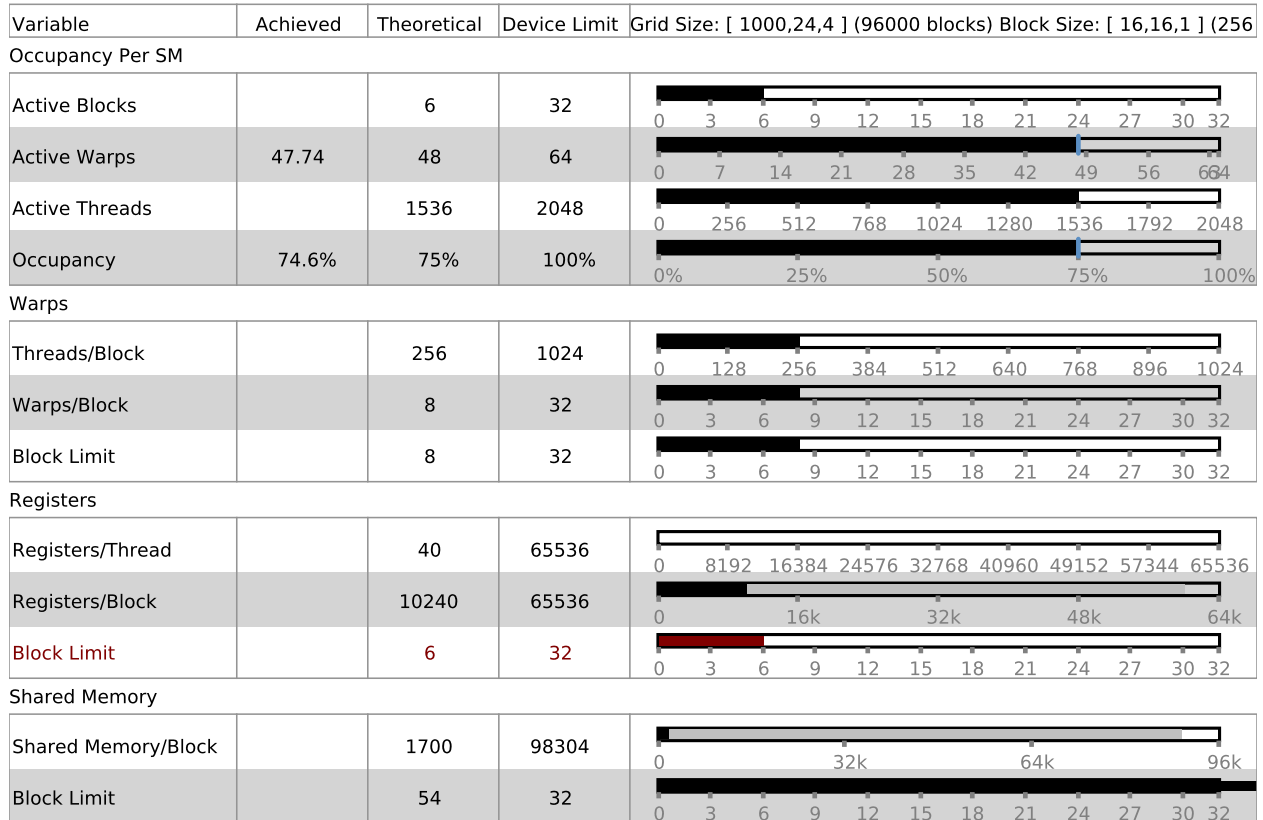
Instruction and memory latency limit the performance of a kernel when the GPU does not have enough work to keep busy. The performance of latency-limited kernels can often be improved by increasing occupancy. Occupancy is a measure of how many warps the kernel has active on the GPU, relative to the maximum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

### 4.1. GPU Utilization May Be Limited By Register Usage

Theoretical occupancy is less than 100% but is large enough that increasing occupancy may not improve performance. You can attempt the following optimization to increase the number of warps on each SM but it may not lead to increased performance.

The kernel uses 40 registers for each thread (10240 registers for each block). This register usage is likely preventing the kernel from fully utilizing the GPU. Device "TITAN V" provides up to 65536 registers for each block. Because the kernel uses 10240 registers for each block each SM is limited to simultaneously executing 6 blocks (48 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.*

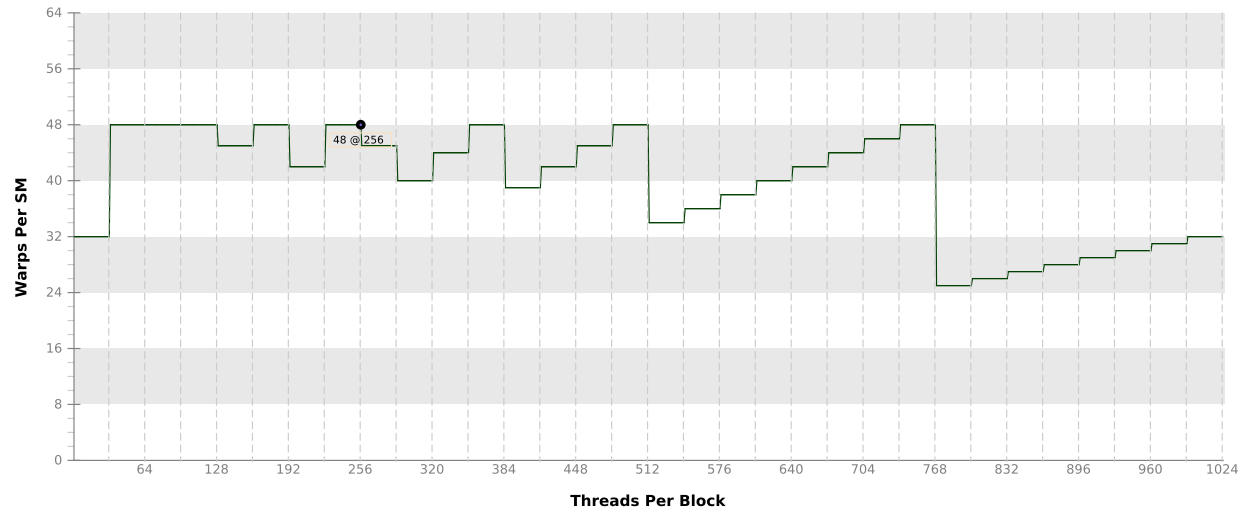


### 4.2. Occupancy Charts

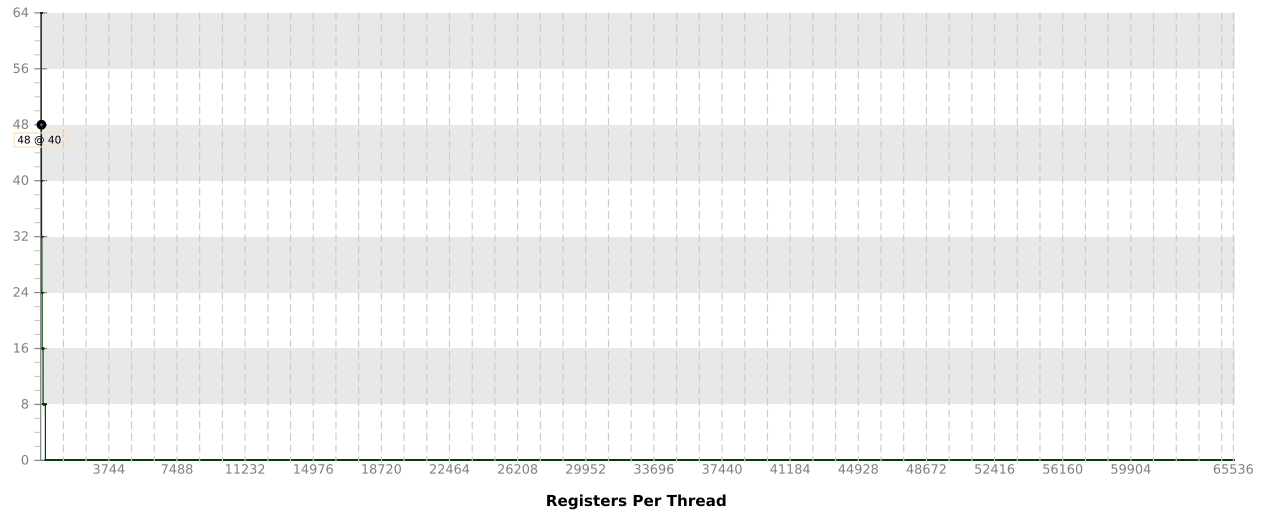
The following charts show how varying different components of the kernel will impact theoretical occupancy.

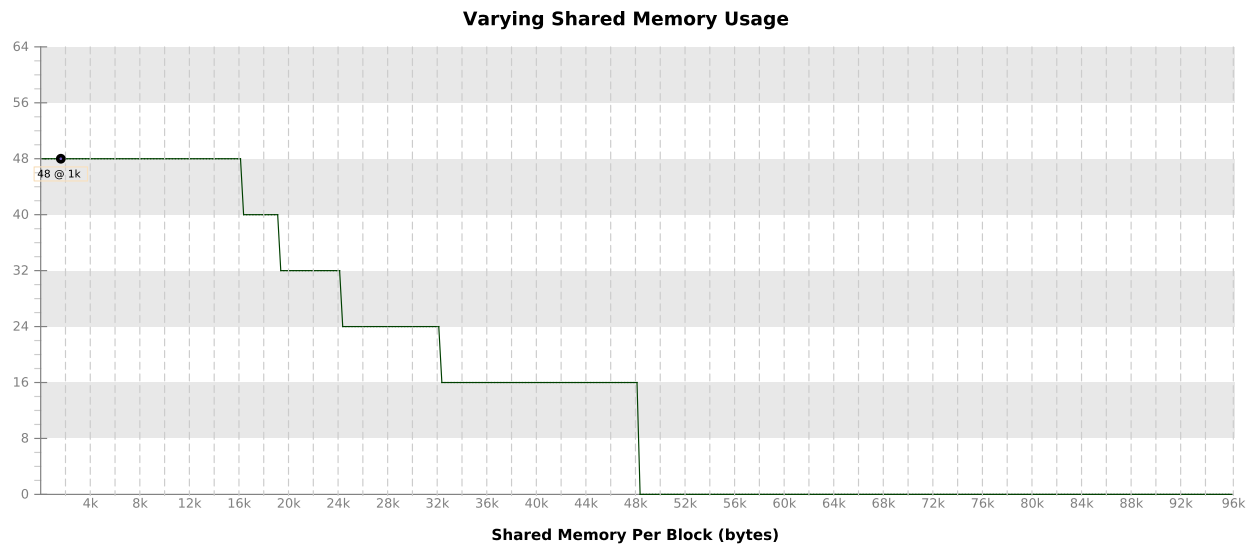


**Varying Block Size**



**Varying Register Count**





### 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.

