Analysis Report

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

Duration	28.65153 ms (28,651,534 ns)
Grid Size	[273,1,10000]
Block Size	[64,1,1]
Registers/Thread	54
Shared Memory/Block	256 B
Shared Memory Executed	0 B
Shared Memory Bank Size	4 B

[0] TITAN V

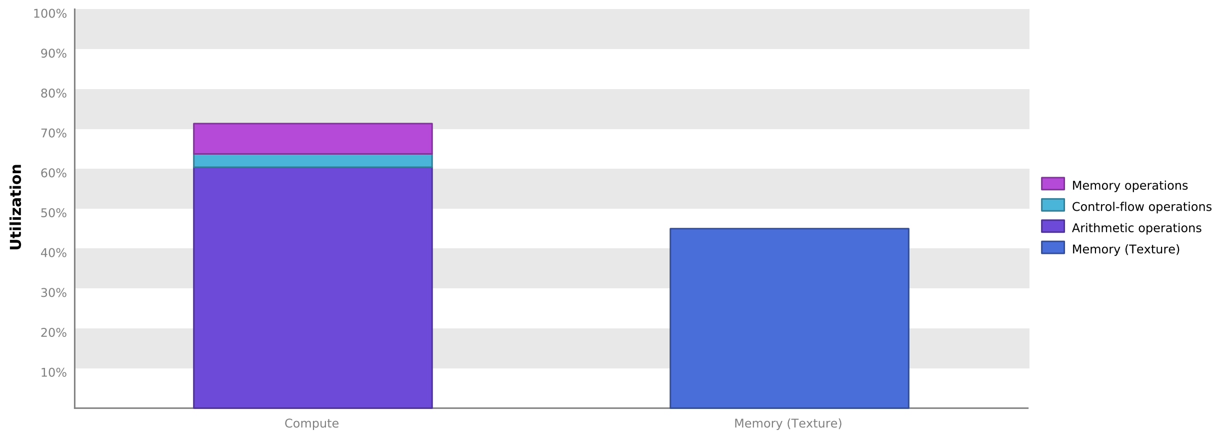
GPU UUID	GPU-b8a50260-383e-be56-a455-6f0653b61e6d
Compute Capability	7.0
Max. Threads per Block	1024
Max. Threads per Multiprocessor	2048
Max. Shared Memory per Block	48 KiB
Max. Shared Memory per Multiprocessor	96 KiB
Max. Registers per Block	65536
Max. Registers per Multiprocessor	65536
Max. Grid Dimensions	[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::register_tiled_f..." 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. Kernel Profile - Instruction Execution

The Kernel Profile - Instruction Execution shows the execution count, inactive threads, and predicated threads for each source and assembly line of the kernel. Using this information you can pinpoint portions of your kernel that are making inefficient use of compute resource due to divergence and predication.

Examine portions of the kernel that have high execution counts and inactive or predicated threads to identify optimization opportunities.

Cuda Fuctions :

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

Maximum instruction execution count in assembly: 57330000
Average instruction execution count in assembly: 20035022
Instructions executed for the kernel: 8735270000
Thread instructions executed for the kernel: 258942360000
Non-predicated thread instructions executed for the kernel: 239269060000
Warp non-predicated execution efficiency of the kernel: 85.6%
Warp execution efficiency of the kernel: 92.6%

Source files :

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

2.2. 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.

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

Line 218	Divergence = 0% [0 divergent executions out of 38220000 total executions]
Line 218	Divergence = 0% [0 divergent executions out of 5460000 total executions]
Line 225	Divergence = 7.5% [2850000 divergent executions out of 38220000 total executions]
Line 233	Divergence = 50% [19110000 divergent executions out of 38220000 total executions]
Line 334	Divergence = 50% [2730000 divergent executions out of 5460000 total executions]
Line 336	Divergence = 0% [0 divergent executions out of 2730000 total executions]
Line 339	Divergence = 0% [0 divergent executions out of 2730000 total executions]
Line 342	Divergence = 0% [0 divergent executions out of 2730000 total executions]
Line 345	Divergence = 0% [0 divergent executions out of 2730000 total executions]
Line 348	Divergence = 0% [0 divergent executions out of 2730000 total executions]
Line 351	Divergence = 0% [0 divergent executions out of 2730000 total executions]
Line 354	Divergence = 0% [0 divergent executions out of 2730000 total executions]

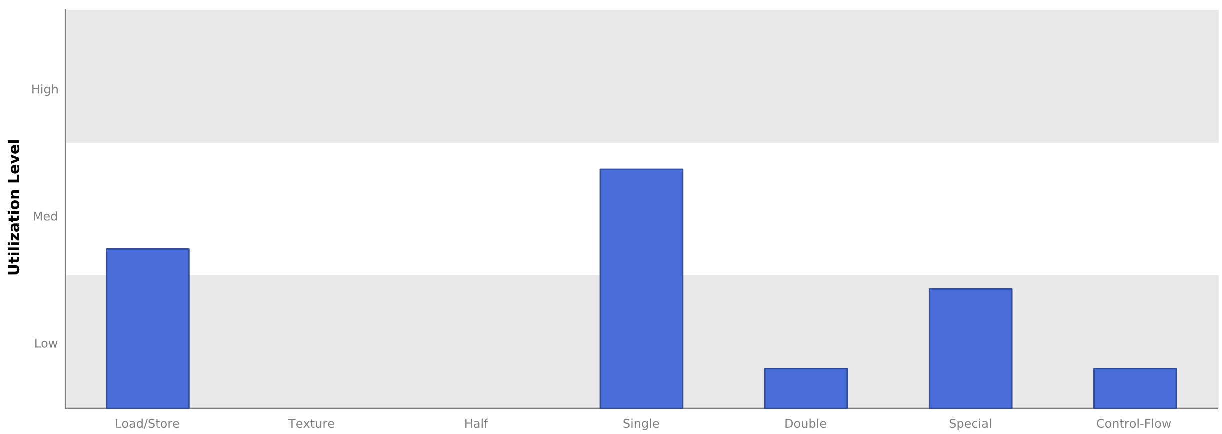
/mxnet/src/operator/custom/./new-forward.cuh

Line 357	Divergence = 0% [0 divergent executions out of 2730000 total executions]
Line 360	Divergence = 0% [0 divergent executions out of 2730000 total executions]
Line 363	Divergence = 0% [0 divergent executions out of 2730000 total executions]
Line 366	Divergence = 0% [0 divergent executions out of 2730000 total executions]
Line 369	Divergence = 0% [0 divergent executions out of 2730000 total executions]
Line 372	Divergence = 0% [0 divergent executions out of 2730000 total executions]
Line 375	Divergence = 0% [0 divergent executions out of 2730000 total executions]
Line 378	Divergence = 0% [0 divergent executions out of 2730000 total executions]
Line 381	Divergence = 0% [0 divergent executions out of 2730000 total executions]
Line 389	Divergence = 0% [0 divergent executions out of 2720000 total executions]

2.3. Function Unit Utilization

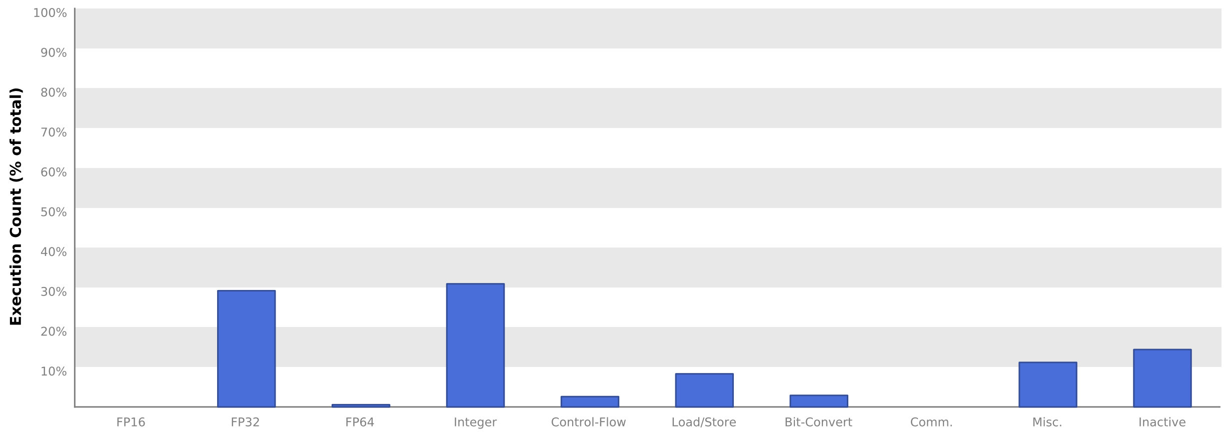
Different types of instructions are executed on different function units within each SM. Performance can be limited if a function unit is over-used by the instructions executed by the kernel. The following results show that the kernel's performance is not limited by overuse of any function unit.

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.



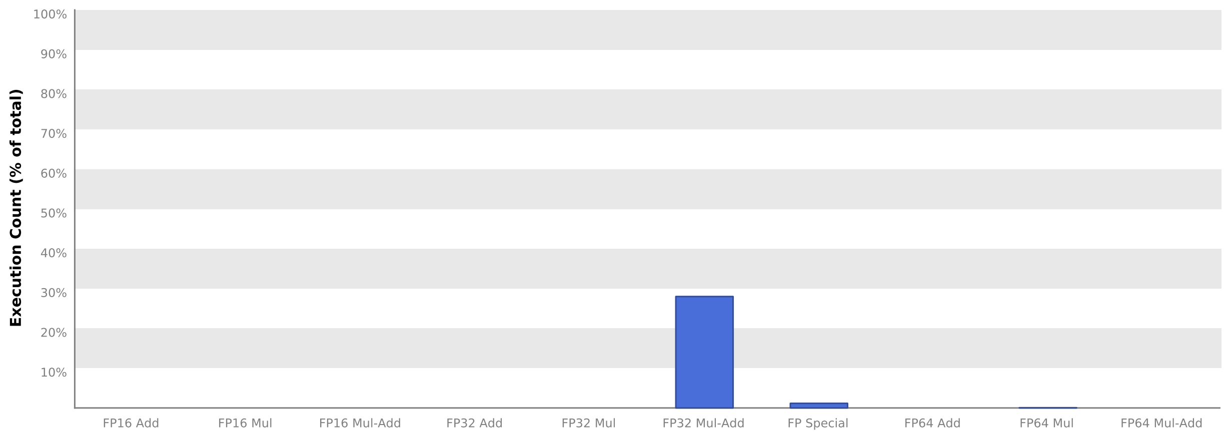
2.4. Instruction Execution Counts

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.



2.5. 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.



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.

3.1. 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	1249409906	5,581.707 GB/s	
Shared Stores	38347961	171.319 GB/s	
Shared Total	1287757867	5,753.026 GB/s	Idle Low Medium High Max
L2 Cache			
Reads	41976911	46.883 GB/s	
Writes	522720038	583.81 GB/s	
Total	564696949	630.692 GB/s	Idle Low Medium High Max
Unified Cache			
Local Loads	0	0 B/s	
Local Stores	0	0 B/s	
Global Loads	941354530	1,051.369 GB/s	
Global Stores	522720000	583.81 GB/s	
Texture Reads	1479356509	6,608.988 GB/s	
Unified Total	2943431039	8,244.167 GB/s	Idle Low Medium High Max
Device Memory			
Reads	6128748	6.845 GB/s	
Writes	65339882	72.976 GB/s	
Total	71468630	79.821 GB/s	Idle Low Medium High Max
System Memory			
[PCIe configuration: Gen3 x16, 8 Gbit/s]			
Reads	0	0 B/s	Idle Low Medium High Max
Writes	5	5.584 kB/s	Idle Low Medium High Max

3.2. 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.

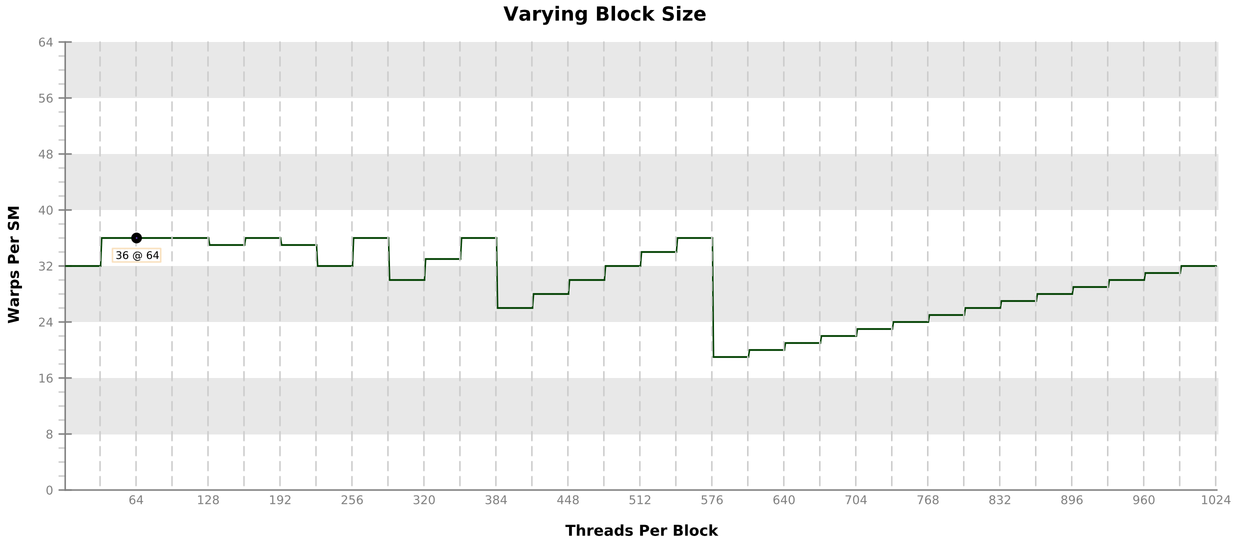
The kernel uses 54 registers for each thread (3456 registers for each block). This register usage is likely preventing the kernel from fully utilizing the GPU. Device "TITAN V" provides up to 65536 registers for each block. Because the kernel uses 3456 registers for each block each SM is limited to simultaneously executing 18 blocks (36 warps). Chart "Varying Register Count" below shows how changing register usage will change the number of blocks that can execute on each SM.

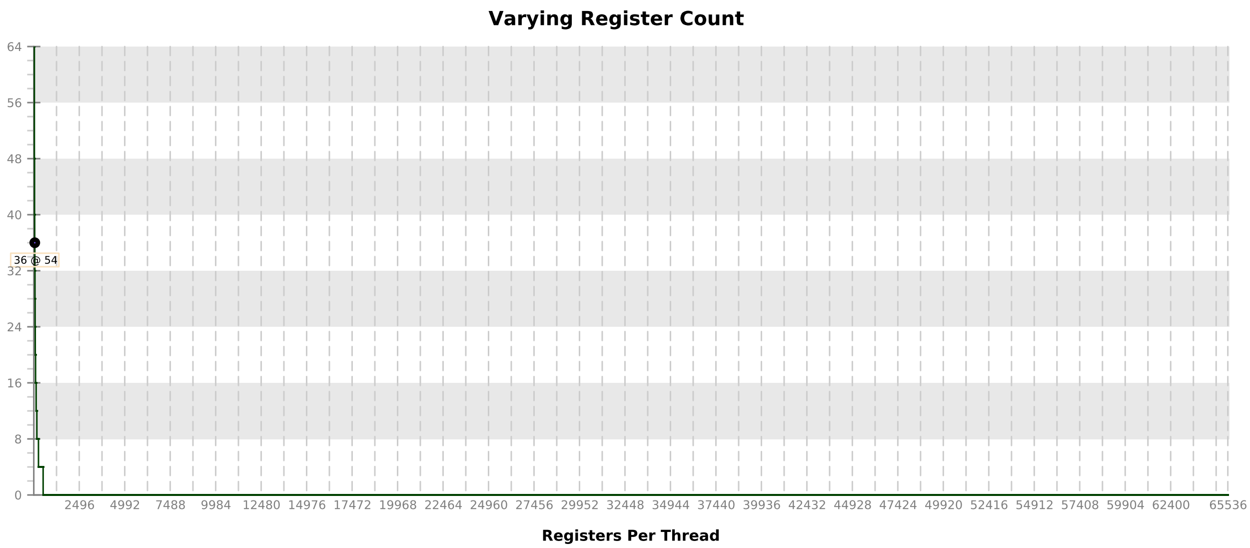
Optimization: Use the -maxrregcount flag or the __launch_bounds__ qualifier to decrease the number of registers used by each thread. This will increase the number of blocks that can execute on each SM. On devices with Compute Capability 5.2 turning global cache off can increase the occupancy limited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

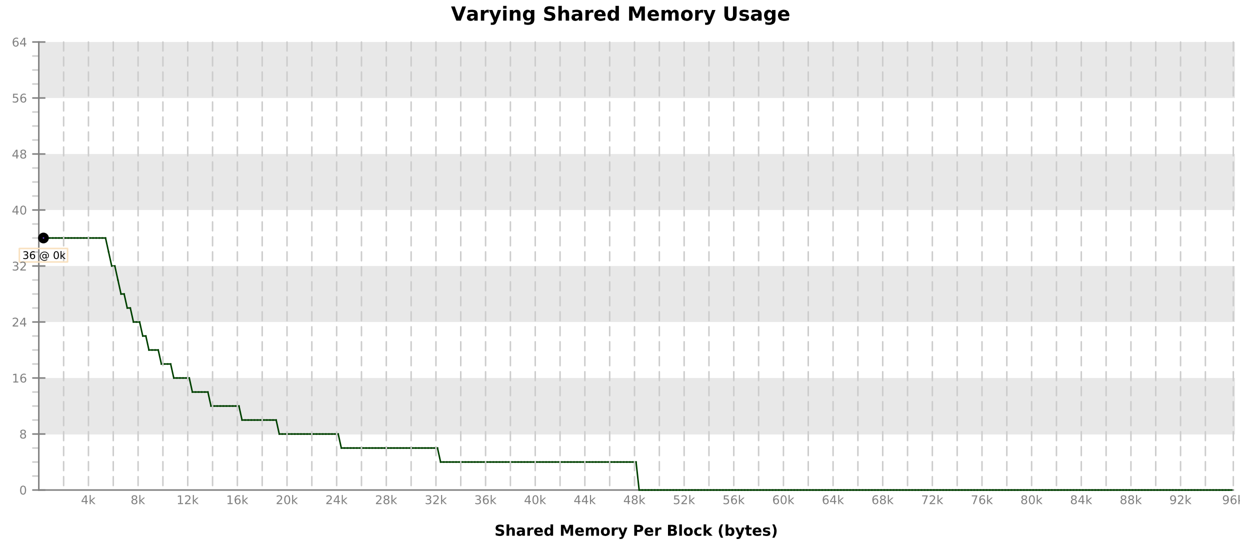
Variable	Achieved	Theoretical	Device Limit	Grid Size: [273,1,10000] (2730000 blocks) Block Size: [64,1,1] ((
Occupancy Per SM				
Active Blocks		18	32	0 3 6 9 12 15 18 21 24 27 30 32
Active Warps	33.4	36	64	0 7 14 21 28 35 42 49 56 63 64
Active Threads		1152	2048	0 256 512 768 1024 1280 1536 1792 2048
Occupancy	52.2%	56.2%	100%	0% 25% 50% 75% 100%
Warps				
Threads/Block		64	1024	0 128 256 384 512 640 768 896 1024
Warps/Block		2	32	0 3 6 9 12 15 18 21 24 27 30 32
Block Limit		32	32	0 3 6 9 12 15 18 21 24 27 30 32
Registers				
Registers/Thread		54	65536	0 8192 16384 24576 32768 40960 49152 57344 65536
Registers/Block		3584	65536	0 16k 32k 48k 64k
Block Limit		18	32	0 3 6 9 12 15 18 21 24 27 30 32
Shared Memory				
Shared Memory/Block		256	98304	0 32k 64k 96k
Block Limit		384	32	0 3 6 9 12 15 18 21 24 27 30 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
36 @ 64
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
36 @ 54
2496
4992
7488
9984
12480
14976
17472
19968
22464
24960
27456
29952
32448
34944
37440
39936
42432
44928
47424
49920
52416
54912
57408
59904
62400
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.