Analysis Report

distanceKernel(uchar4*, int, int, int2)

Duration	30.00096 ms (30,000,959 ns)
Grid Size	[64,64,1]
Block Size	[32,32,1]
Registers/Thread	36
Shared Memory/Block	0 B
Shared Memory Requested	96 KiB
Shared Memory Executed	96 KiB
Shared Memory Bank Size	4 B

[0] GeForce GTX 1070

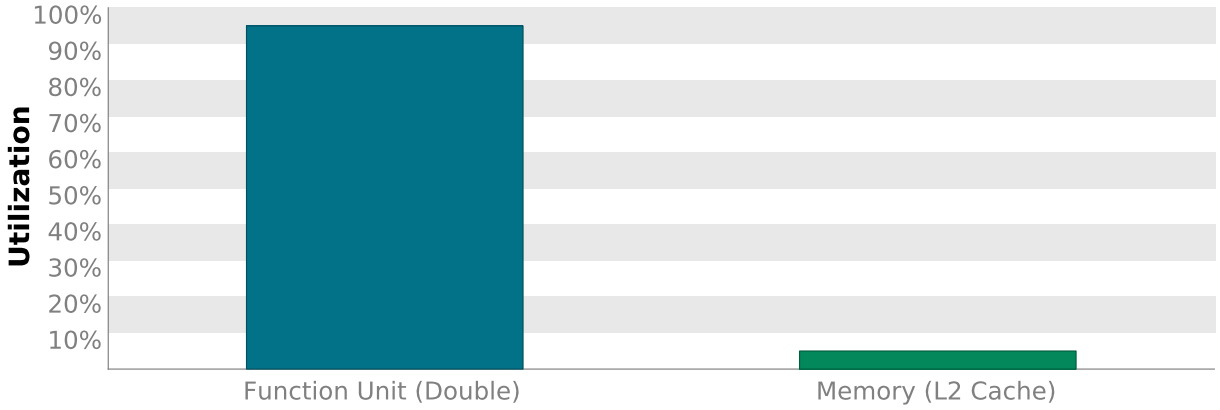
GPU UUID	GPU-71a27d7b-cdad-e200-a709-2a830b569027
Compute Capability	6.1
Max. Threads per Block	1024
Max. Threads per Multiprocessor	2048
Max. Shared Memory per Block	48 KiB
Max. Shared Memory per Multiprocessor	96 KiB
Max. Registers per Block	65536
Max. Registers per Multiprocessor	65536
Max. Grid Dimensions	[2147483647, 65535, 65535]
Max. Block Dimensions	[1024, 1024, 64]
Max. Warps per Multiprocessor	64
Max. Blocks per Multiprocessor	32
Half Precision FLOP/s	53.91 GigaFLOP/s
Single Precision FLOP/s	6.9 TeraFLOP/s
Double Precision FLOP/s	215.64 GigaFLOP/s
Number of Multiprocessors	15
Multiprocessor Clock Rate	1.797 GHz
Concurrent Kernel	true
Max IPC	6
Threads per Warp	32
Global Memory Bandwidth	256.256 GB/s
Global Memory Size	7.926 GiB
Constant Memory Size	64 KiB
L2 Cache Size	2 MiB
Memcpy Engines	2
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 "distanceKernel" 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 "GeForce GTX 1070" 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 instructions do not overuse a function unit. The results below indicate that compute performance may be limited by overuse of a function unit.

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 :

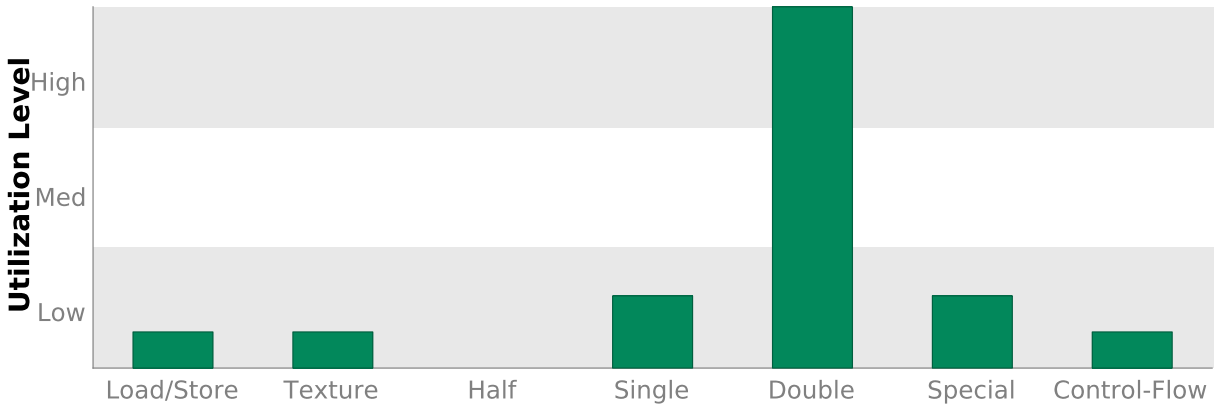
distanceKernel(uchar4*, int, int, int2)

Maximum instruction execution count in assembly: 1966080
Average instruction execution count in assembly: 399834
Instructions executed for the kernel: 530180601
Thread instructions executed for the kernel: 16957656470
Non-predicated thread instructions executed for the kernel: 15590855104
Warp non-predicated execution efficiency of the kernel: 91.9%
Warp execution efficiency of the kernel: 100.0%

2.2. GPU Utilization Is Limited By Function Unit Usage

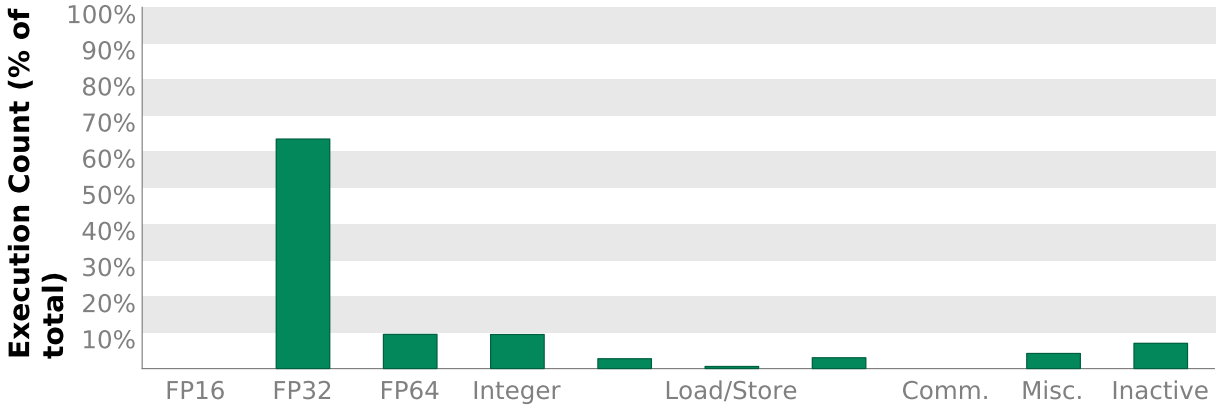
Different types of instructions are executed on different function units within each SM. Performance can be limited if a function unit is over-used by the instructions executed by the kernel. The following results show that the kernel's performance is potentially limited by overuse of the following function units: Double.

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.



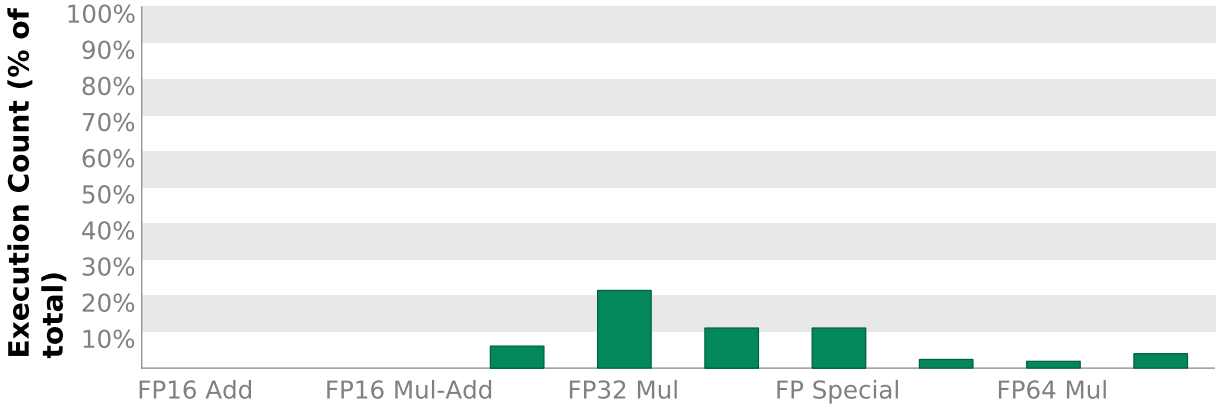
2.3. Instruction Execution Counts

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.

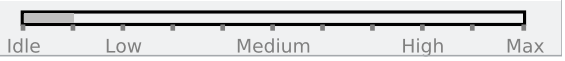
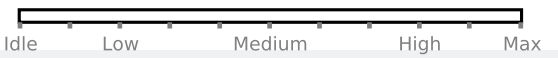


3. Memory Bandwidth

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.

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	0	0 B/s	
Shared Stores	0	0 B/s	
Shared Total	0	0 B/s	
L2 Cache			
Reads	1047005	1.854 GB/s	
Writes	3145768	5.57 GB/s	
Total	4192773	7.424 GB/s	
Unified Cache			
Local Loads	3	5.312 kB/s	
Local Stores	1572864	2.785 GB/s	
Global Loads	2	0 B/s	
Global Stores	1572864	2.785 GB/s	
Texture Reads	3	5.312 kB/s	
Unified Total	3145736	5.57 GB/s	
Device Memory			
Reads	1130590	2.002 GB/s	
Writes	529471	937.552 MB/s	
Total	1660061	2.94 GB/s	
System Memory			
[PCIe configuration: Gen3 x16, 8 Gbit/s]			
Reads	0	0 B/s	
Writes	5	8.853 kB/s	

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 results below indicate that the GPU does not have enough work because instruction execution is stalling excessively.

4.1. Kernel Profile - PC Sampling

The Kernel Profile - PC Sampling gives the number of samples for each source and assembly line with various stall reasons. The samples are collected at a period of 128 [2^7] cycles. You can change the period under Settings->Analysis tab. The allowed values are from 5 to 31. Increasing the period would reduce the number of samples collected.

Using this information you can pinpoint portions of your kernel that are introducing latencies and the reason for the latency. Samples are taken in round robin order for all active warps at a fixed number of cycles regardless of whether the warp is issuing an instruction or not.

Instruction Issued - Warp was issued

Instruction Fetch - The next assembly instruction has not yet been fetched.

Execution Dependency - An input required by the instruction is not yet available. Execution dependency stalls can potentially be reduced by increasing instruction-level parallelism.

Memory Dependency - A load/store cannot be made because the required resources are not available or are fully utilized, or too many requests of a given type are outstanding. Data request stalls can potentially be reduced by optimizing memory alignment and access patterns.

Texture - The texture sub-system is fully utilized or has too many outstanding requests.

Synchronization - The warp is blocked at a __syncthreads() call.

Constant - A constant load is blocked due to a miss in the constants cache.

Pipe Busy - The compute resource(s) required by the instruction is not yet available.

Memory Throttle - Large number of pending memory operations prevent further forward progress. These can be reduced by combining several memory transactions into one.

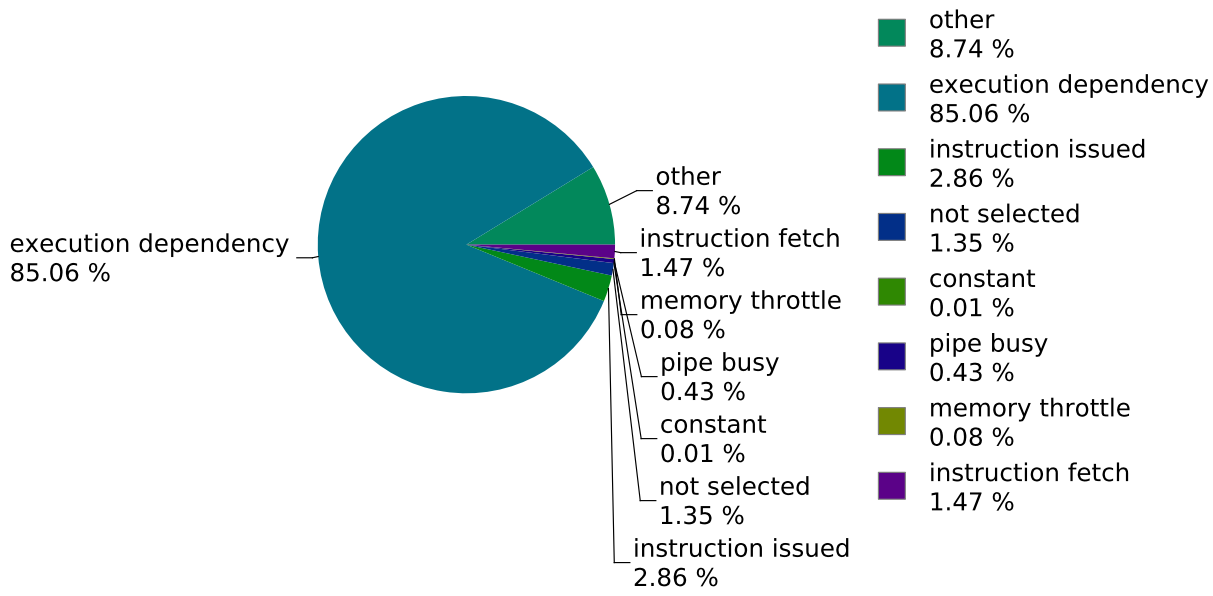
Not Selected - Warp was ready to issue, but some other warp issued instead. You may be able to sacrifice occupancy without impacting latency hiding and doing so may help improve cache hit rates.

Other - The warp is blocked for an uncommon reason.

Sleeping -The warp is blocked, yielded or sleeping.

Examine portions of the kernel that have high number of samples to know where the maximum time was spent and observe the latency reasons for those samples to identify optimization opportunities.

Cuda Functions	Sample Count	% of Kernel Samples
distanceKernel(uchar4*, int, int, int2)	4146692	100.0



4.2. 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.

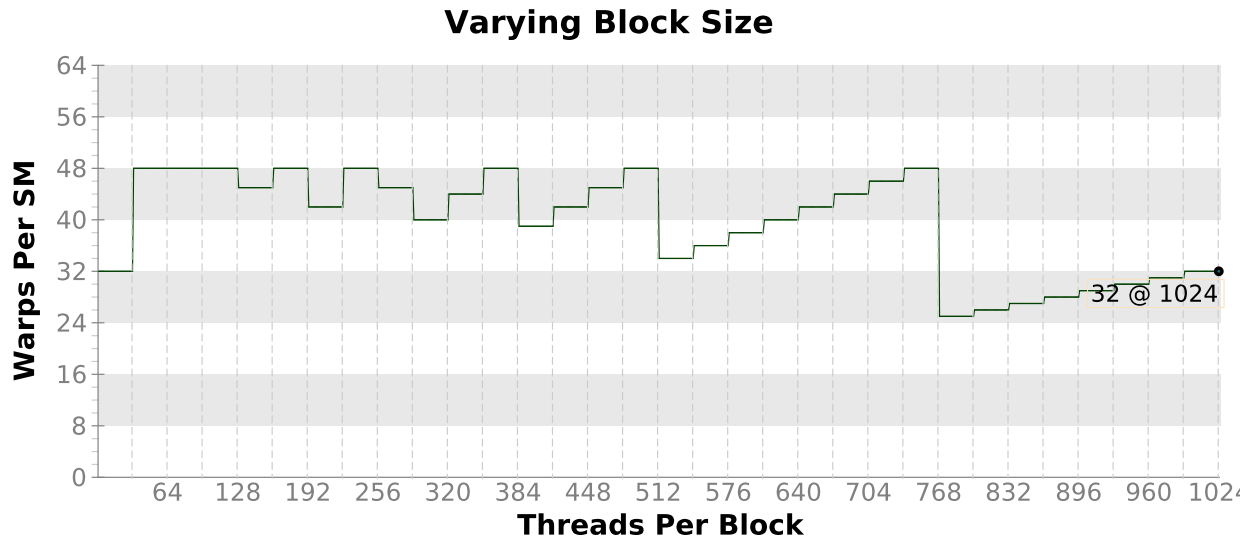
The kernel uses 36 registers for each thread (36864 registers for each block). This register usage is likely preventing the kernel from fully utilizing the GPU. Device "GeForce GTX 1070" provides up to 65536 registers for each block. Because the kernel uses 36864 registers for each block each SM is limited to simultaneously executing 1 block (32 warps). Chart "Varying Register Count" below shows how changing register usage will change the number of blocks that can execute on each SM.

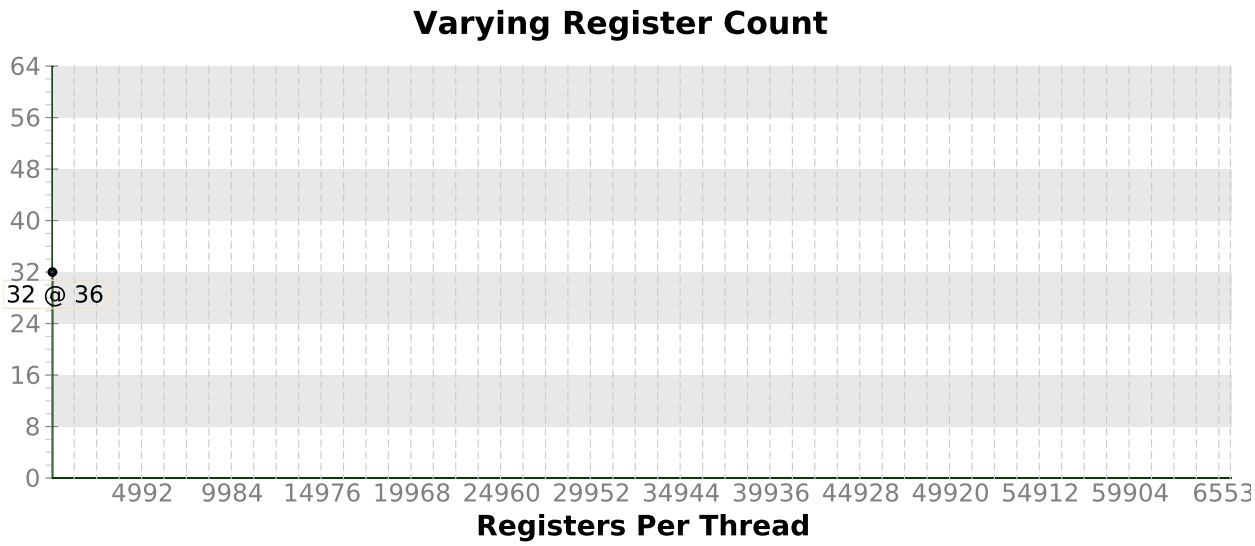
Optimization: Use the -maxrregcount flag or the __launch_bounds__ qualifier to decrease the number of registers used by each thread. This will increase the number of blocks that can execute on each SM. On devices with Compute Capability 5.2 turning global cache off can increase the occupancy limited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

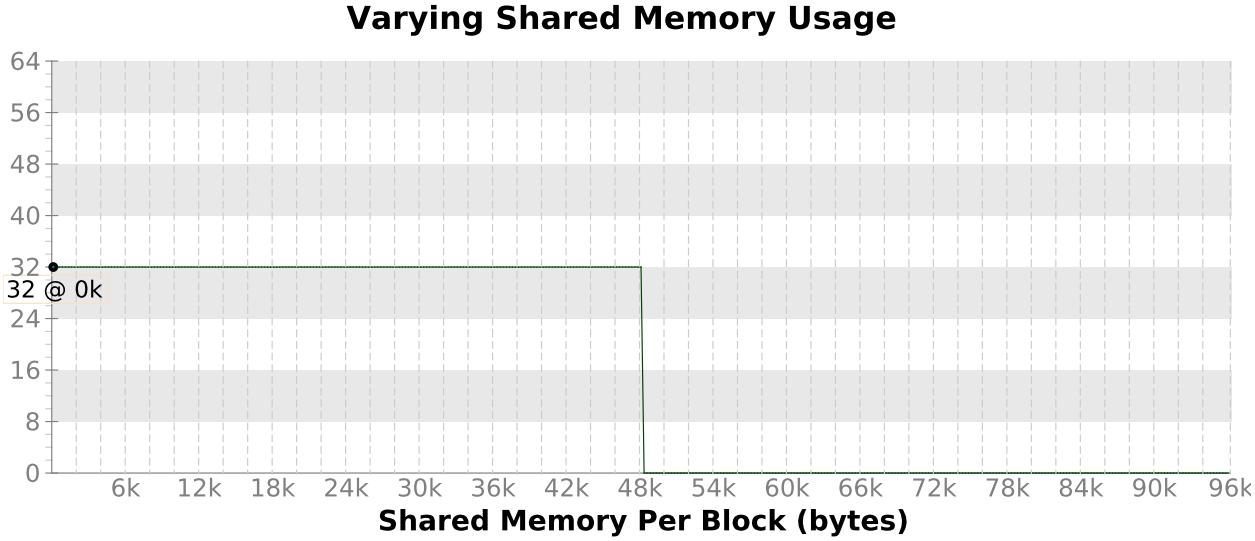
Variable	Achieved	Theoretical	Device Limit	Grid Size: [64,64,1] (4096 blocks) Block Size: [32,32,1
Occupancy Per SM				
Active Blocks		1	32	0 4 8 12 16 20 24 28 32
Active Warps	31.7	32	64	0 9 18 27 36 45 54 64
Active Threads		1024	2048	0 512 1024 1536 2048
Occupancy	49.5%	50%	100%	0% 25% 50% 75% 100%
Warps				
Threads/Block		1024	1024	0 256 512 768 1024
Warps/Block		32	32	0 4 8 12 16 20 24 28 32
Block Limit		2	32	0 4 8 12 16 20 24 28 32
Registers				
Registers/Thread		36	65536	0 16384 32768 49152 65536
Registers/Block		40960	65536	0 16k 32k 48k 64k
Block Limit		1	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

4.3. Occupancy Charts

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




Varying Register Count
64
56
48
40
32
24
16
8
0
32 @ 36
4992
9984
14976
19968
24960
29952
34944
39936
44928
49920
54912
59904
6553
Registers Per Thread


Varying Shared Memory Usage
64
56
48
40
32
24
16
8
0
32 @ 0k
6k
12k
18k
24k
30k
36k
42k
48k
54k
60k
66k
72k
78k
84k
90k
96k
Shared Memory Per Block (bytes)