Analysis Report

mult_kernel_compressed_data(unsigned char*, unsigned char*, unsigned char*, int, int, int, int)

Duration	755.406 µs
Grid Size	[32,32,1]
Block Size	[32,32,1]
Registers/Thread	32
Shared Memory/Block	16.25 KiB
Shared Memory Executed	32.5 KiB
Shared Memory Bank Size	4 B

[0] GeForce GTX 980 Ti

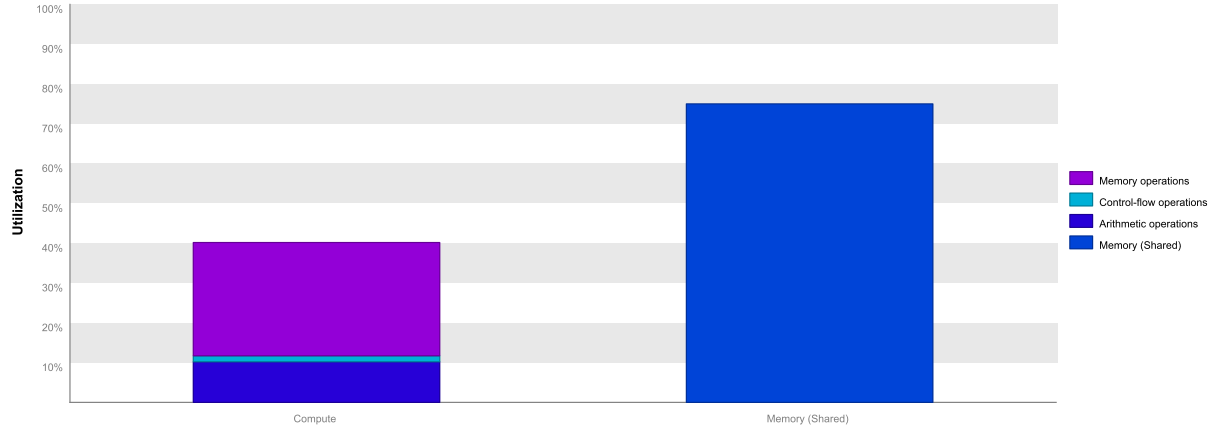
GPU UUID	GPU-1efff26a-ba19-4be3-1414-34841cb941c3
Compute Capability	5.2
Max. Threads per Block	1024
Max. Threads per Multiprocessor	2048
Max. Shared Memory per Block	48 KiB
Max. Shared Memory per Multiprocessor	96 KiB
Max. Registers per Block	65536
Max. Registers per Multiprocessor	65536
Max. Grid Dimensions	[2147483647, 65535, 65535]
Max. Block Dimensions	[1024, 1024, 64]
Max. Warps per Multiprocessor	64
Max. Blocks per Multiprocessor	32
Single Precision FLOP/s	7.271 TeraFLOP/s
Double Precision FLOP/s	227.216 GigaFLOP/s
Number of Multiprocessors	22
Multiprocessor Clock Rate	1.291 GHz
Concurrent Kernel	true
Max IPC	6
Threads per Warp	32
Global Memory Bandwidth	336.48 GB/s
Global Memory Size	5.94 GiB
Constant Memory Size	64 KiB
L2 Cache Size	3 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 "mult_kernel_compressed_data" 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 "GeForce GTX 980 Ti" 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 Shared 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 shared memory.

2.1. 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	12582912	2,410.819 GB/s	
Shared Stores	1310720	251.127 GB/s	
Shared Total	13893632	2,661.946 GB/s	Idle Low Medium High Max
L2 Cache			
Reads	5245329	251.244 GB/s	
Writes	32774	1.57 GB/s	
Total	5278103	252.814 GB/s	Idle Low Medium High Max
Unified Cache			
Local Loads	0	0 B/s	
Local Stores	0	0 B/s	
Global Loads	8388610	251.127 GB/s	
Global Stores	32768	1.57 GB/s	
Texture Reads	8388608	401.803 GB/s	
Unified Total	16809986	654.5 GB/s	Idle Low Medium High Max
Device Memory			
Reads	45219	2.166 GB/s	
Writes	13664	654.487 MB/s	
Total	58883	2.82 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	239.493 kB/s	Idle Low Medium High Max

2.2. 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.

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

3.1. 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 256 [2^8] 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.

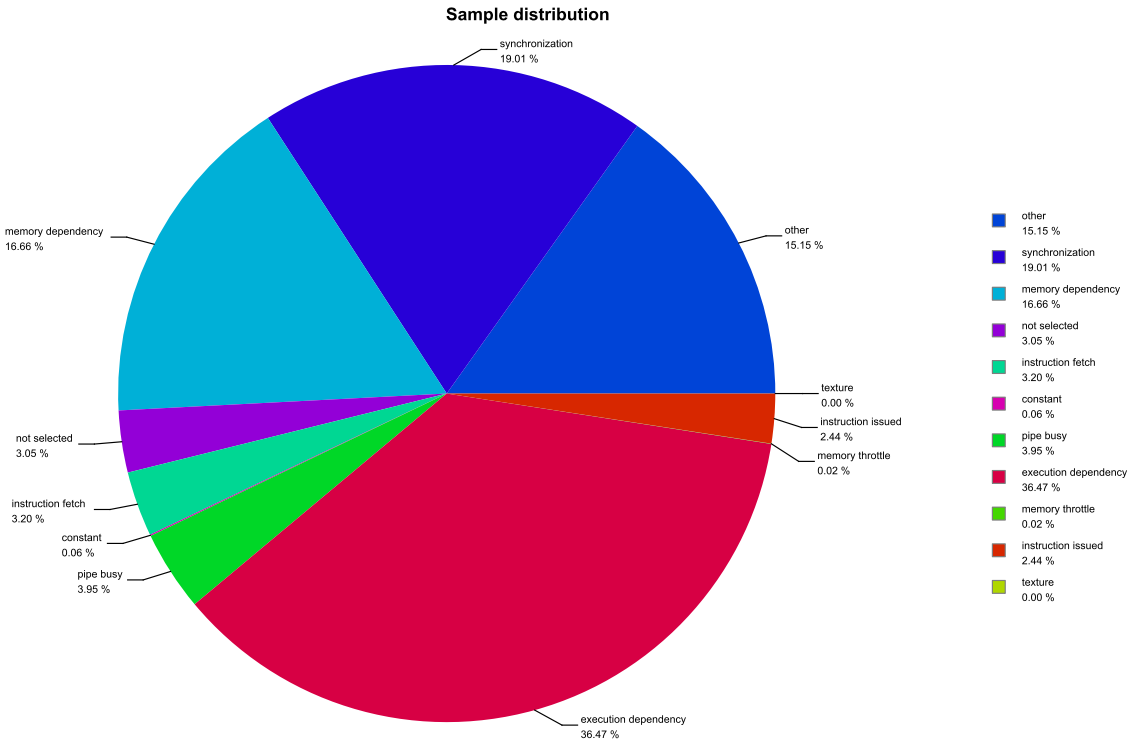
Not Selected - Warp was ready to issue, but some other warp issued instead. You may be able to sacrifice occupancy without impacting latency hiding and doing so may help improve cache hit rates.

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
mult_kernel_compressed_data(unsigned char*, unsigned char*, unsigned char*, int, int, int, int)	75432	100.0



3.2. GPU Utilization Is Limited By Shared Memory Usage

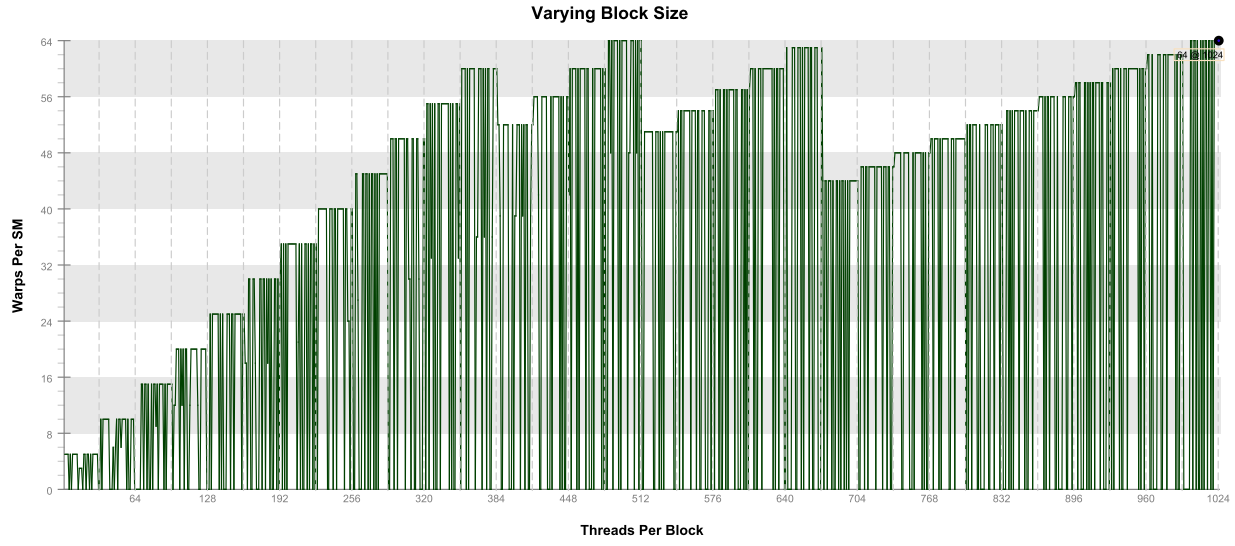
The kernel uses 16.25 KiB of shared memory for each block. This shared memory usage is likely preventing the kernel from fully utilizing the GPU. Device "GeForce GTX 980 Ti" is configured to have 96 KiB of shared memory for each SM. Because the kernel uses 16.25 KiB of shared memory for each block each SM is limited to simultaneously executing 2 blocks (64 warps). Chart "Varying Shared Memory Usage" below shows how changing shared memory usage will change the number of blocks that can execute on each SM.

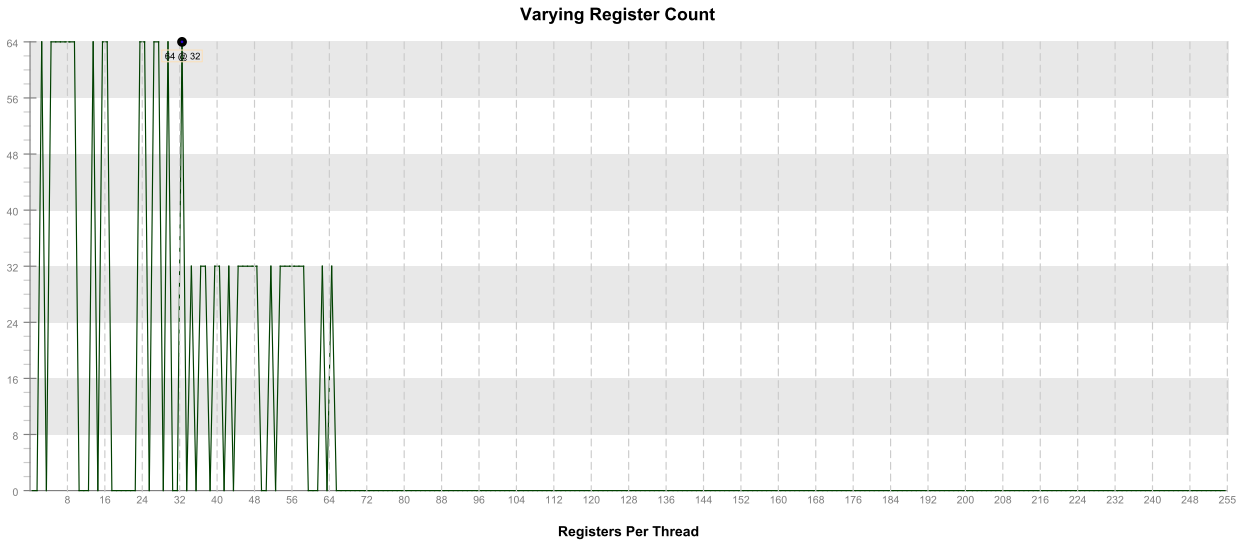
Optimization: Reduce shared memory usage to increase the number of blocks that can execute on each SM. You can also increase the number of blocks that can execute on each SM by increasing the amount of shared memory available to your kernel. You do this by setting the preferred cache configuration to "prefer shared".

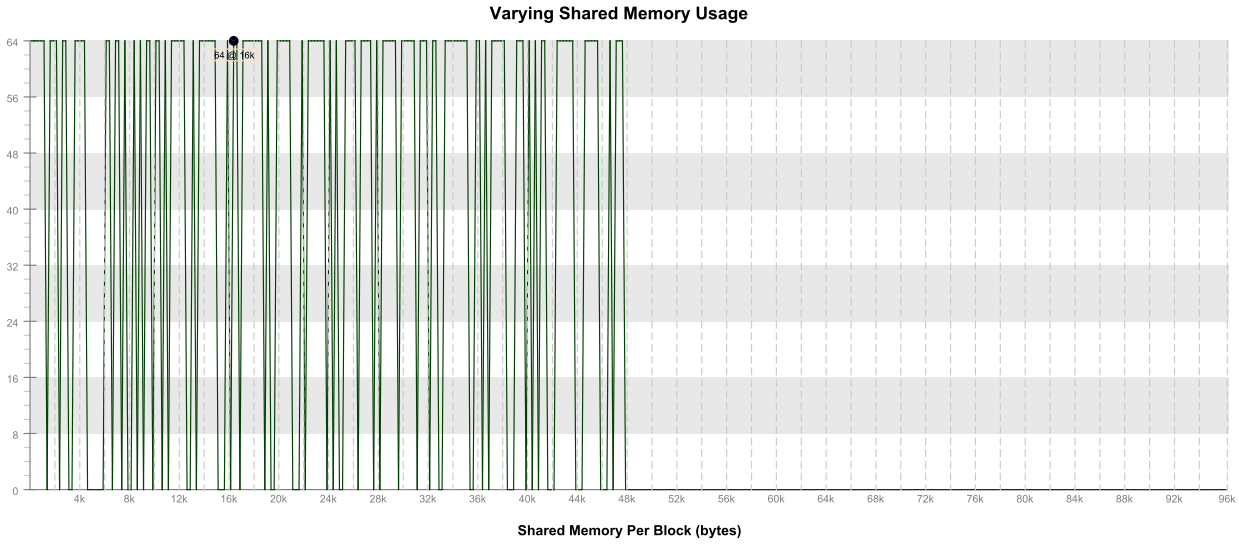
Variable	Achieved	Theoretical	Device Limit	Grid Size: [32,32,1] (1024 blocks) Block Size: [32,32,1] (1024 threads)
Occupancy Per SM				
Active Blocks		0	32	0 3 6 9 12 15 18 21 24 27 30 32
Active Warps	63.17	0	64	0 6 12 18 24 30 36 42 48 54 60 64
Active Threads		0	2048	0 256 512 768 1024 1280 1536 1792 2048
Occupancy	98.7%	0%	100%	0% 25% 50% 75% 100%
Warps				
Threads/Block		1024	1024	0 128 256 384 512 640 768 896 1024
Warps/Block		32	32	0 3 6 9 12 15 18 21 24 27 30 32
Block Limit		2	32	0 3 6 9 12 15 18 21 24 27 30 32
Registers				
Registers/Thread		32	255	0 32 64 96 128 160 192 224 255
Registers/Block		32768	65536	0 16k 32k 48k 64k
Block Limit		2	32	0 3 6 9 12 15 18 21 24 27 30 32
Shared Memory				
Shared Memory/Block		16640	98304	0 16k 32k 48k 64k 80k 96k
Block Limit		0	32	0 3 6 9 12 15 18 21 24 27 30 32

3.3. Occupancy Charts

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

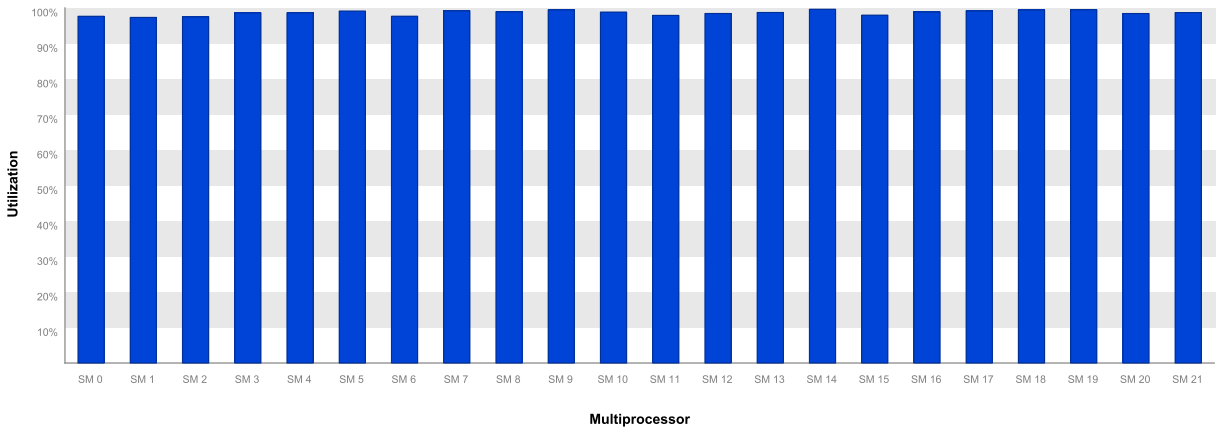






3.4. Multiprocessor Utilization

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.


100%
90%
80%
70%
60%
50%
40%
30%
20%
10%
Utilization
SM 0
SM 1
SM 2
SM 3
SM 4
SM 5
SM 6
SM 7
SM 8
SM 9
SM 10
SM 11
SM 12
SM 13
SM 14
SM 15
SM 16
SM 17
SM 18
SM 19
SM 20
SM 21
Multiprocessor

4. Compute Resources

GPU compute resources limit the performance of a kernel when those resources are insufficient or poorly utilized.

4.1. 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 :

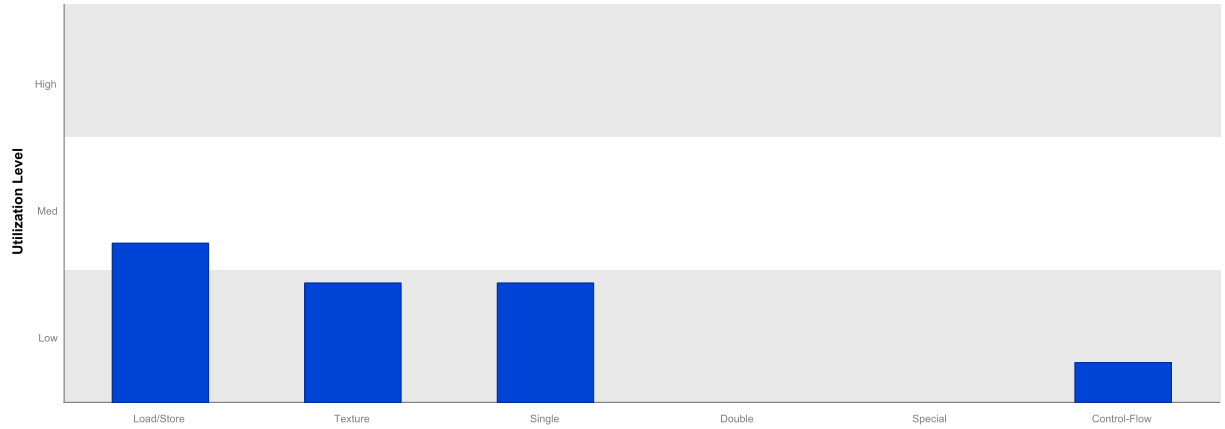
mult_kernel_compressed_data(unsigned char*, unsigned char*, unsigned char*, int, int, int, int)

Maximum instruction execution count in assembly: 262144
Average instruction execution count in assembly: 166640
Instructions executed for the kernel: 38993920
Thread instructions executed for the kernel: 1247805440
Non-predicated thread instructions executed for the kernel: 1227856240
Warp non-predicated execution efficiency of the kernel: 98.4%
Warp execution efficiency of the kernel: 100.0%

4.2. 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.

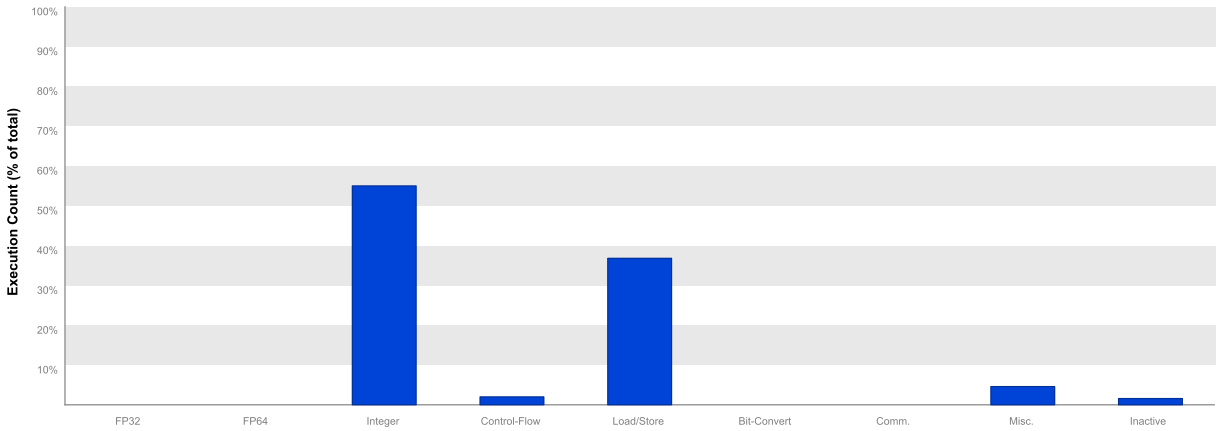
Load/Store - Load and store instructions for shared and constant memory.
Texture - Load and store instructions for local, global, and texture memory.
Single - Single-precision integer and floating-point arithmetic instructions.
Double - Double-precision floating-point arithmetic instructions.
Special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



4.3. 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.



4.4. 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.