Analysis Report

monochrome(pixel const *, pixel*, int)

Duration	276.177 µs
Grid Size	[2110,1,1]
Block Size	[128,1,1]
Registers/Thread	9
Shared Memory/Block	0 B
Shared Memory Executed	0 B
Shared Memory Bank Size	4 B

[0] GeForce GT 1030

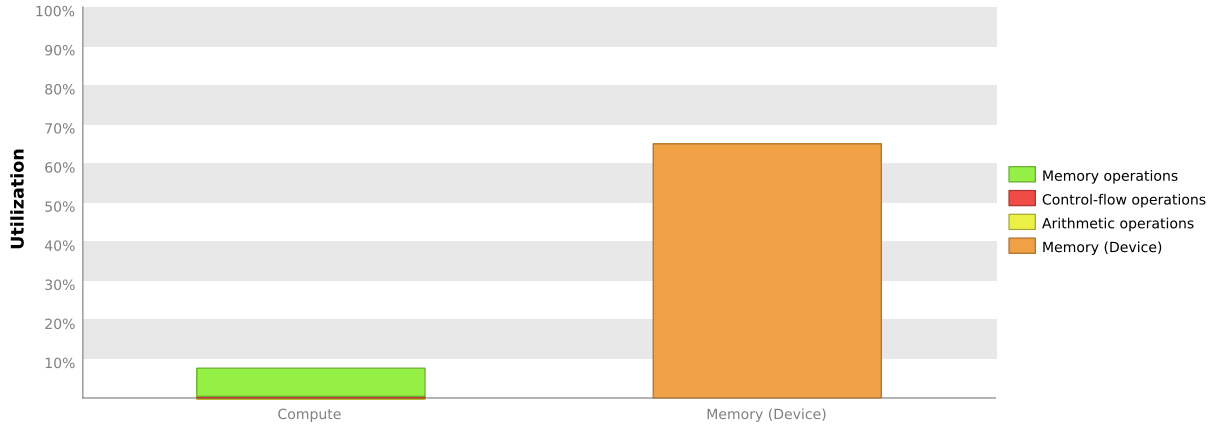
GPU UUID	GPU-85939596-8af0-a29c-4ab3-11659ad2611c
Compute Capability	6.1
Max. Threads per Block	1024
Max. Threads per Multiprocessor	2048
Max. Shared Memory per Block	48 KiB
Max. Shared Memory per Multiprocessor	96 KiB
Max. Registers per Block	65536
Max. Registers per Multiprocessor	65536
Max. Grid Dimensions	[2147483647, 65535, 65535]
Max. Block Dimensions	[1024, 1024, 64]
Max. Warps per Multiprocessor	64
Max. Blocks per Multiprocessor	32
Half Precision FLOP/s	8.808 GigaFLOP/s
Single Precision FLOP/s	1.127 TeraFLOP/s
Double Precision FLOP/s	35.232 GigaFLOP/s
Number of Multiprocessors	3
Multiprocessor Clock Rate	1.468 GHz
Concurrent Kernel	true
Max IPC	6
Threads per Warp	32
Global Memory Bandwidth	48.064 GB/s
Global Memory Size	1.954 GiB
Constant Memory Size	64 KiB
L2 Cache Size	512 KiB
Memcpy Engines	2
PCIe Generation	2
PCIe Link Rate	5 Gbit/s
PCIe Link Width	4

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 "monochrome" 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 GT 1030" 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 Device 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 device memory.

2.1. 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.

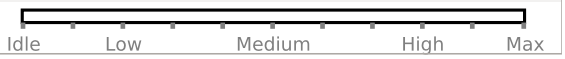
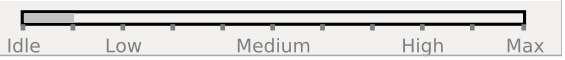
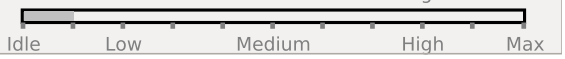
/home/hpjeon/hw/class/udemy_CUDA10/sec3.11/monochrome.cu

Line 17	Global Load L2 Transactions/Access = 16, Ideal Transactions/Access = 4 [135000 L2 transactions for 8438 total executions]
Line 17	Global Load L2 Transactions/Access = 16, Ideal Transactions/Access = 4 [135000 L2 transactions for 8438 total executions]
Line 17	Global Load L2 Transactions/Access = 16, Ideal Transactions/Access = 4 [135000 L2 transactions for 8438 total executions]
Line 20	Global Store L2 Transactions/Access = 16, Ideal Transactions/Access = 4 [135000 L2 transactions for 8438 total executions]
Line 21	Global Store L2 Transactions/Access = 16, Ideal Transactions/Access = 4 [135000 L2 transactions for 8438 total executions]
Line 22	Global Store L2 Transactions/Access = 16, Ideal Transactions/Access = 4 [135000 L2 transactions for 8438 total executions]
Line 23	Global Load L2 Transactions/Access = 16, Ideal Transactions/Access = 4 [135000 L2 transactions for 8438 total executions]
Line 23	Global Store L2 Transactions/Access = 16, Ideal Transactions/Access = 4 [135000 L2 transactions for 8438 total executions]

2.2. 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	540040	62.547 GB/s	
Writes	540013	62.544 GB/s	
Total	1080053	125.09 GB/s	
Unified Cache			
Local Loads	0	0 B/s	
Local Stores	0	0 B/s	
Global Loads	540002	62.542 GB/s	
Global Stores	540000	62.542 GB/s	
Texture Reads	135000	15.636 GB/s	
Unified Total	1215002	140.72 GB/s	
Device Memory			
Reads	135006	15.636 GB/s	
Writes	135802	15.728 GB/s	
Total	270808	31.365 GB/s	
System Memory			
[PCIe configuration: Gen2 x4, 5 Gbit/s]			
Reads	0	0 B/s	
Writes	5	579.093 kB/s	

2.3. 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.

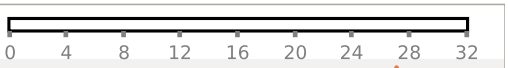
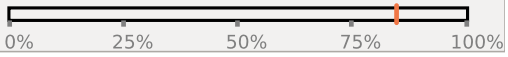
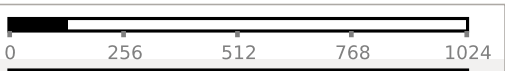
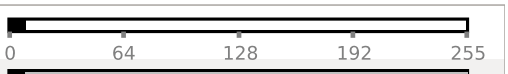
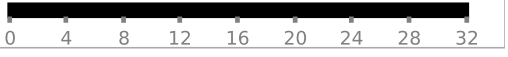
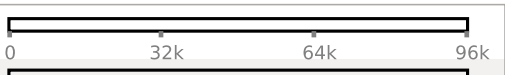
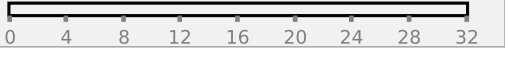
3. Instruction and Memory Latency

Instruction and memory latency limit the performance of a kernel when the GPU does not have enough work to keep busy. The 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 amount of shared memory used by the kernel.

3.1. GPU Utilization Is Limited By Shared Memory Usage

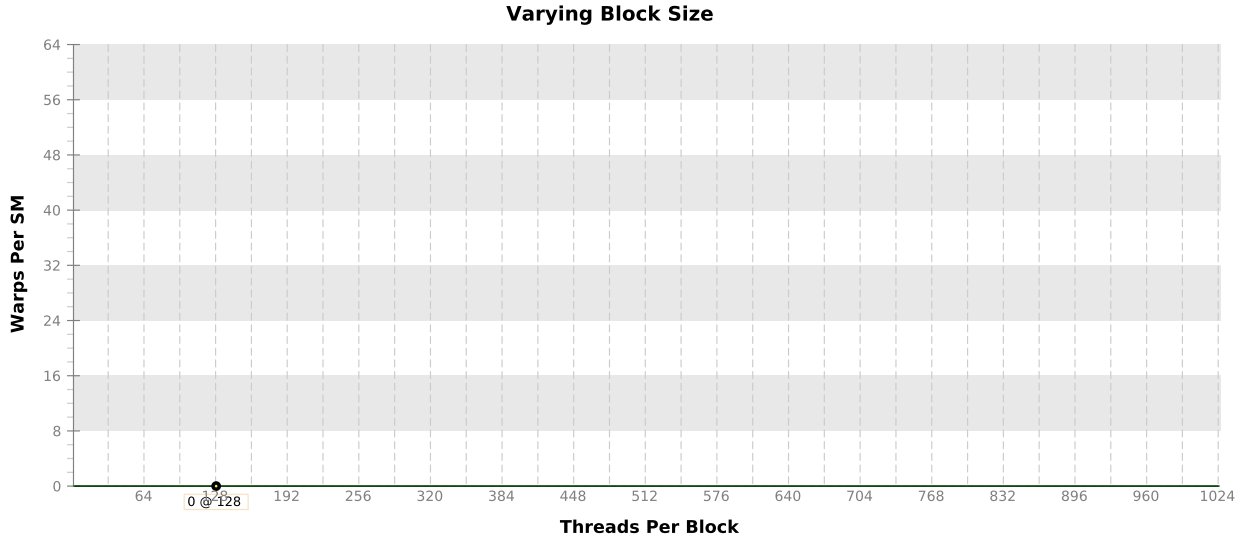
The kernel uses 0 B of shared memory for each block. This shared memory usage is likely preventing the kernel from fully utilizing the GPU. Device "GeForce GT 1030" is configured to have 96 KiB of shared memory for each SM. Because the kernel uses 0 B of shared memory for each block each SM is limited to simultaneously executing 16 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.

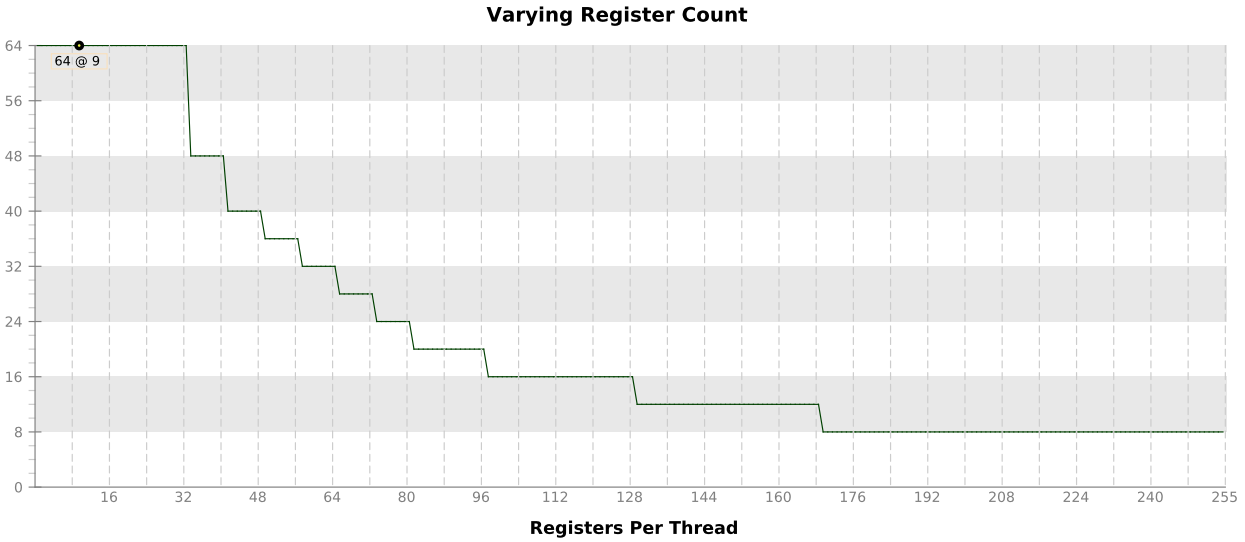
Optimization: Reduce shared memory usage to increase the number of blocks that can execute on each SM. You can also increase the number of blocks that can execute on each SM by increasing the amount of shared memory available to your kernel. You do this by setting the preferred cache configuration to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

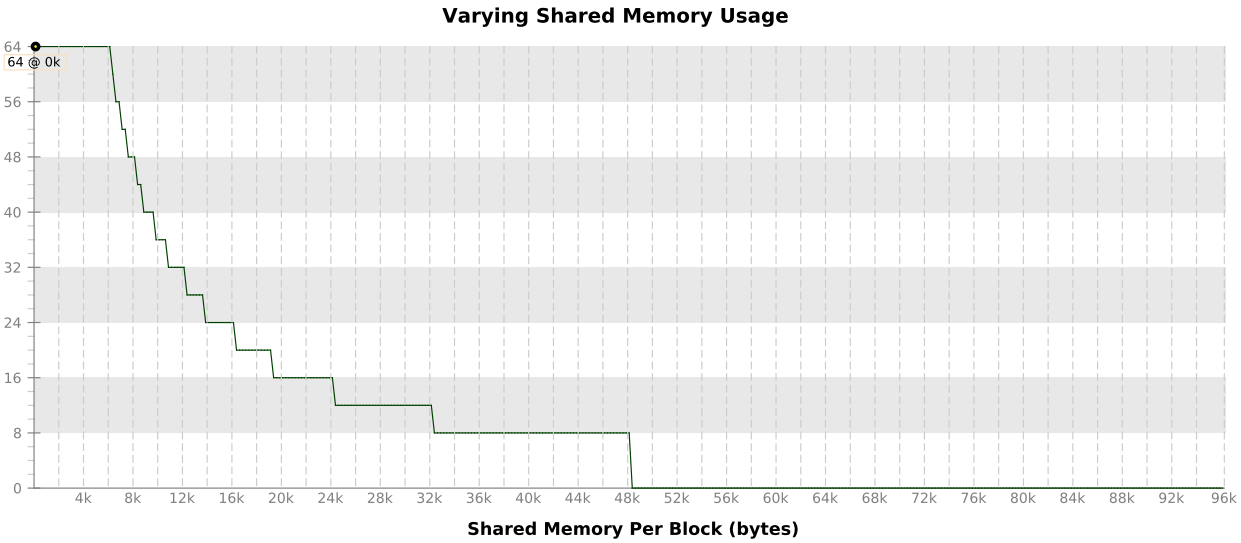
Variable	Achieved	Theoretical	Device Limit	Grid Size: [2110,1,1] (2110 blocks) Block Size: [128,1,
Occupancy Per SM				
Active Blocks		0	32	
Active Warps	53.84	0	64	
Active Threads		0	2048	
Occupancy	84.1%	0%	100%	
Warps				
Threads/Block		128	1024	
Warps/Block		4	32	
Block Limit		16	32	
Registers				
Registers/Thread		9	255	
Registers/Block		2048	65536	
Block Limit		32	32	
Shared Memory				
Shared Memory/Block		0	98304	
Block Limit		0	32	

3.2. Occupancy Charts

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

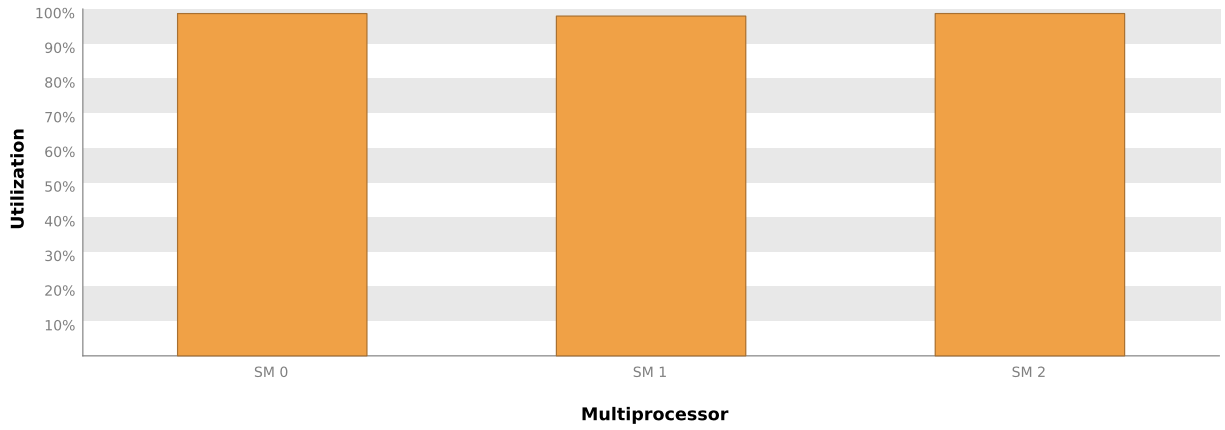

Varying Block Size
Warps Per SM
64
56
48
40
32
24
16
8
0
64
128
192
256
320
384
448
512
576
640
704
768
832
896
960
1024
0 @ 128
Threads Per Block


Varying Register Count
64 @ 9
64
56
48
40
32
24
16
8
0
16
32
48
64
80
96
112
128
144
160
176
192
208
224
240
255
Registers Per Thread



3.3. 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.



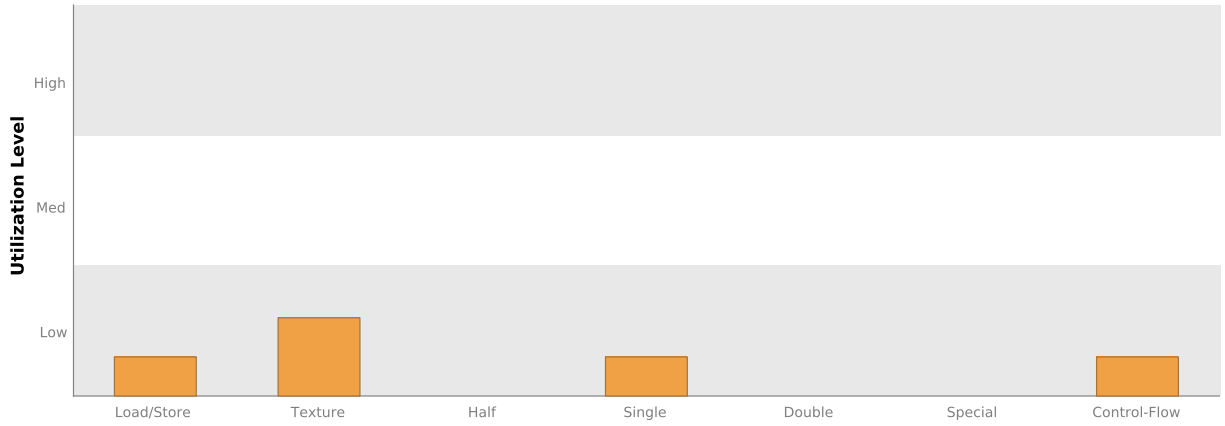
4. Compute Resources

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

4.1. Function Unit Utilization

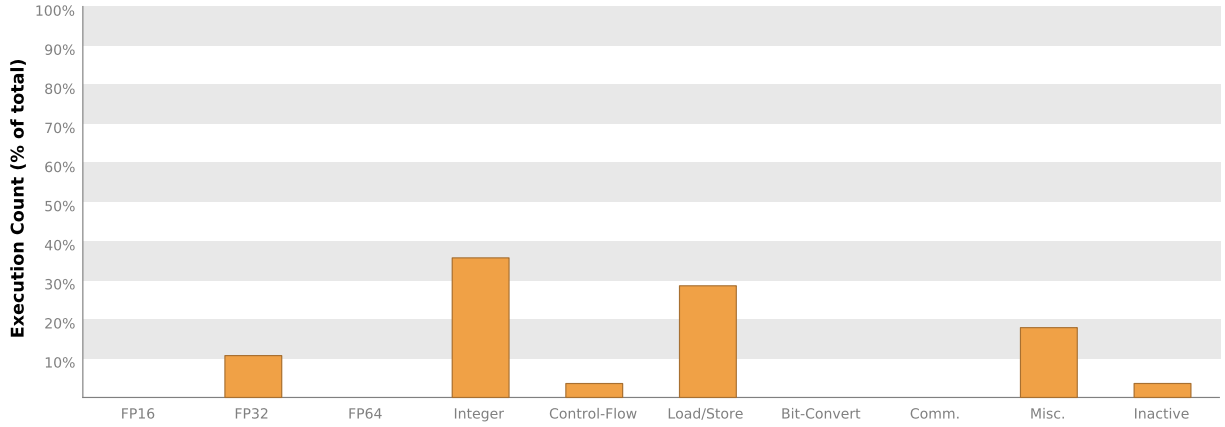
Different types of instructions are executed on different function units within each SM. Performance can be limited if a function unit is over-used by the instructions executed by the kernel. The following results show that the kernel's performance is not limited by overuse of any function unit.

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.



4.2. 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.



4.3. 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.