Analysis Report

trans_kernel(float*, float*, int, int)

Duration	1.366 ms (1,366,404 ns)
Grid Size	[160,480,1]
Block Size	[16,16,1]
Registers/Thread	8
Shared Memory/Block	0 B
Shared Memory Requested	48 KiB
Shared Memory Executed	48 KiB
Shared Memory Bank Size	4 B

[0] Tesla K20c

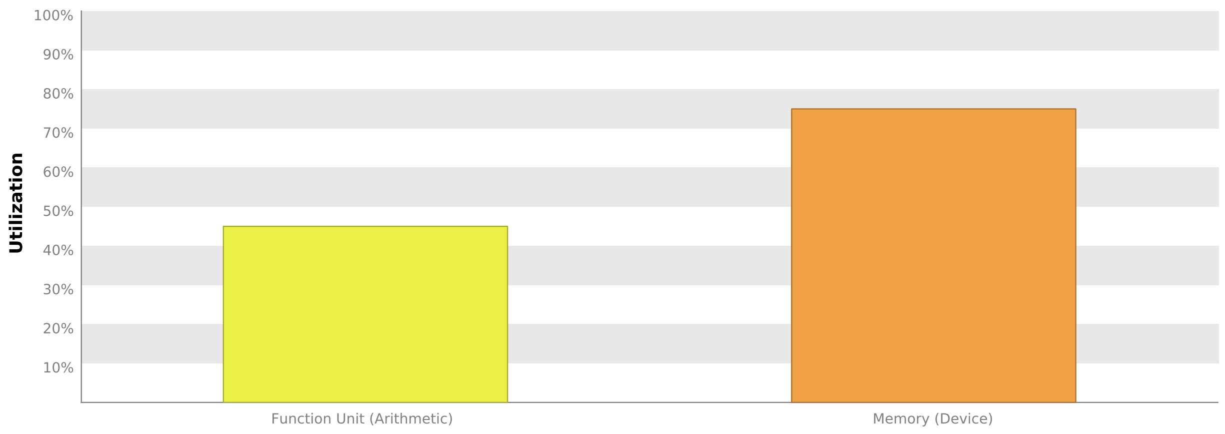
GPU UUID	GPU-5af13825-b086-5bfe-f175-a239faf0f1f9
Compute Capability	3.5
Max. Threads per Block	1024
Max. Shared Memory per Block	48 KiB
Max. Registers per Block	65536
Max. Grid Dimensions	[2147483647, 65535, 65535]
Max. Block Dimensions	[1024, 1024, 64]
Max. Warps per Multiprocessor	64
Max. Blocks per Multiprocessor	16
Single Precision FLOP/s	3.522 TeraFLOP/s
Double Precision FLOP/s	1.174 TeraFLOP/s
Number of Multiprocessors	13
Multiprocessor Clock Rate	705.5 MHz
Concurrent Kernel	true
Max IPC	7
Threads per Warp	32
Global Memory Bandwidth	208 GB/s
Global Memory Size	4.687 GiB
Constant Memory Size	64 KiB
L2 Cache Size	1.25 MiB
Memcpy Engines	2
PCIe Generation	2
PCIe Link Rate	5 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 "trans_kernel" 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 "Tesla K20c" 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. High Local Memory Overhead

Local memory loads and stores account for 66% of total memory traffic. High local memory traffic typically indicates excessive register spilling.

Optimization: Use the -maxrregcount flag or the __launch_bounds__ qualifier to increase the number of registers available to nvcc when compiling the kernel.

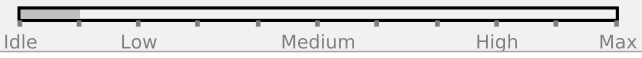
2.2. GPU Utilization Is Limited By Memory Bandwidth

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.

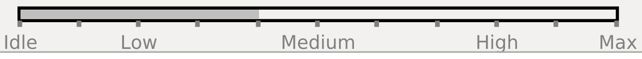
Optimization: Try the following optimizations for the memory with high bandwidth utilization.
L1/Shared Memory - If possible use 64-bit accesses to shared memory and 8-byte bank mode to achieved 2x throughput. Resolve alignment and access pattern issues for global loads and stores.
L2 Cache - Align and block kernel data to maximize L2 cache efficiency.
Texture Cache - Reallocate texture cache data to shared or global memory.
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	

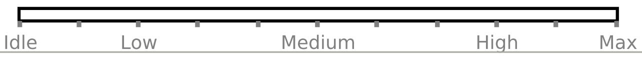
L1/Shared Memory

Local Loads	0	0 B/s	
Local Stores	0	0 B/s	
Shared Loads	0	0 B/s	
Shared Stores	0	0 B/s	
Global Loads	1228800	58.849 GB/s	
Global Stores	2457600	0 B/s	
Atomic	0	0 B/s	
L1/Shared Total	3686400	58.849 GB/s	 Idle Low Medium High Max

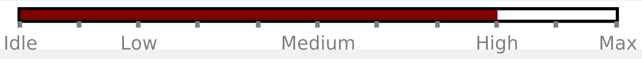
L2 Cache

L1 Reads	2334720	55.906 GB/s	
L1 Writes	4915200	117.697 GB/s	
Texture Reads	0	0 B/s	
Noncoherent Reads	0	0 B/s	
Atomic	0	0 B/s	
Total	7249920	173.603 GB/s	 Idle Low Medium High Max

Texture Cache

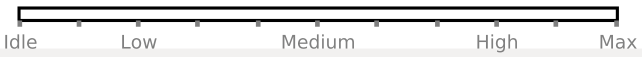
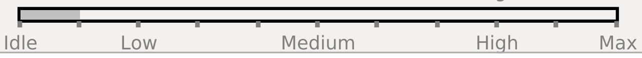
Reads	0	0 B/s	 Idle Low Medium High Max

Device Memory

Reads	3229179	77.324 GB/s	
Writes	2899136	69.421 GB/s	
Total	6128315	146.746 GB/s	 Idle Low Medium High Max
ECC Overhead	1211613	29.013 GB/s	

System Memory

[PCIe configuration: Gen2 x16, 5 Gbit/s]

Reads	0	0 B/s	 Idle Low Medium High Max
Writes	4	95.782 kB/s	 Idle Low Medium High Max

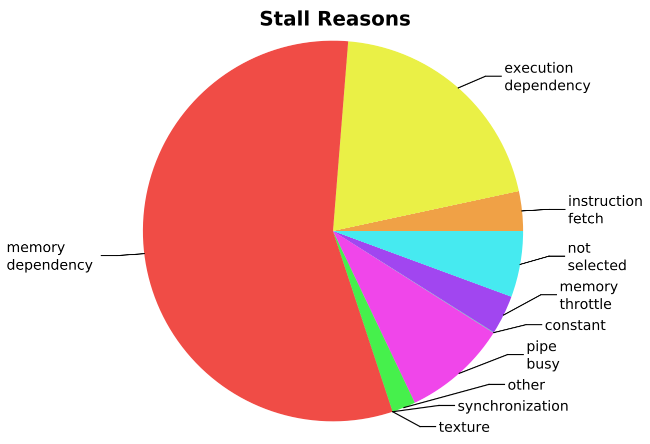
3. Instruction and Memory Latency

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. Instruction Latencies May Be Limiting Performance

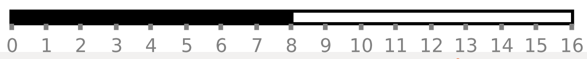
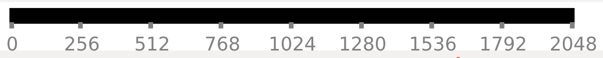
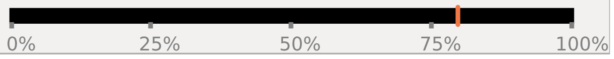
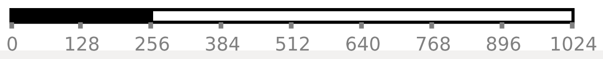
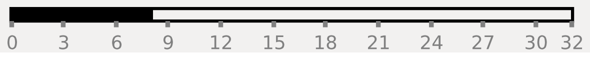
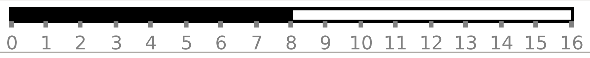
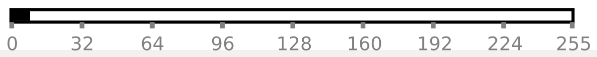
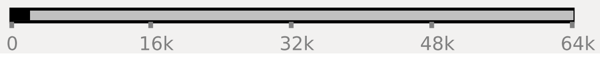
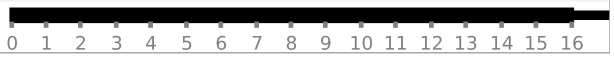
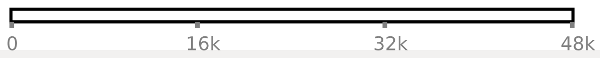
Instruction stall reasons indicate the condition that prevents warps from executing on any given cycle. The following chart shows the break-down of stalls reasons averaged over the entire execution of the kernel. The kernel has good theoretical and achieved occupancy indicating that there are likely sufficient warps executing on each SM. Since occupancy is not an issue it is likely that performance is limited by the instruction stall reasons described below.
Synchronization - The warp is blocked at a __syncthreads() call.
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.
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.
Pipeline Busy - The compute resource(s) required by the instruction is not yet available.
Instruction Fetch - The next assembly instruction has not yet been fetched.
Constant - A constant load is blocked due to a miss in the constants cache.
Texture - The texture sub-system is fully utilized or has too many outstanding requests.

Optimization: Resolve the primary stall issue; memory dependency.



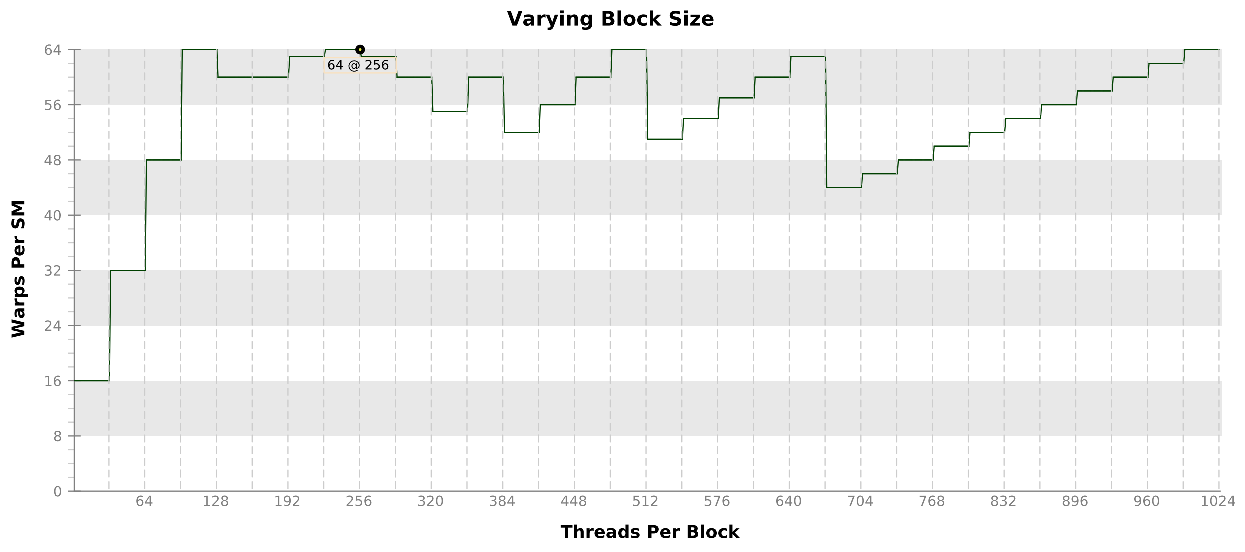
3.2. Occupancy Is Not Limiting Kernel Performance

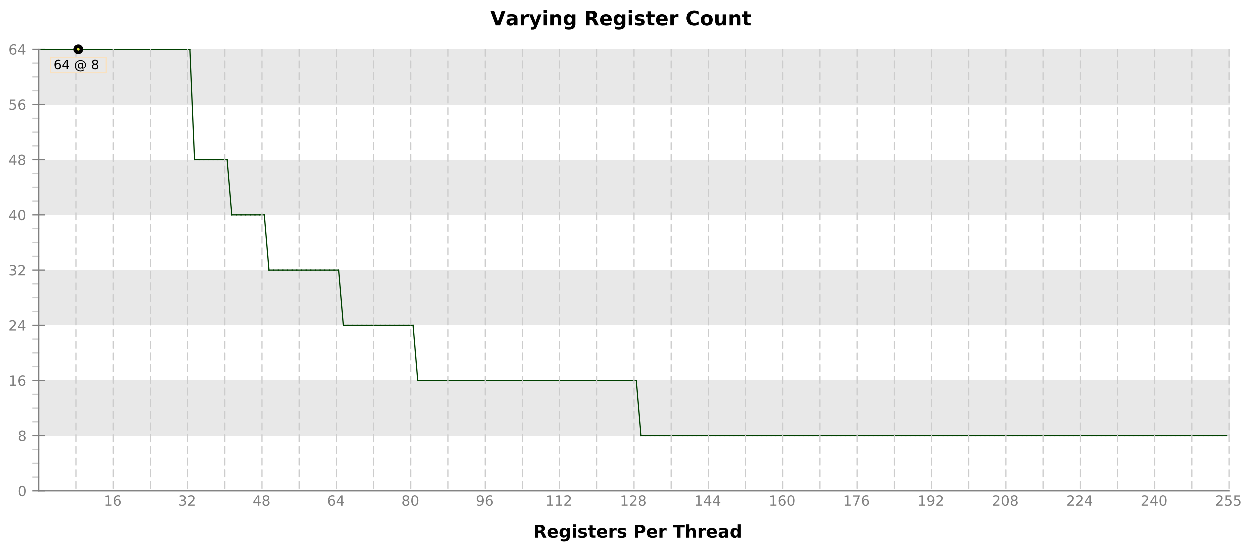
The kernel's block size, register usage, and shared memory usage allow it to fully utilize all warps on the GPU.

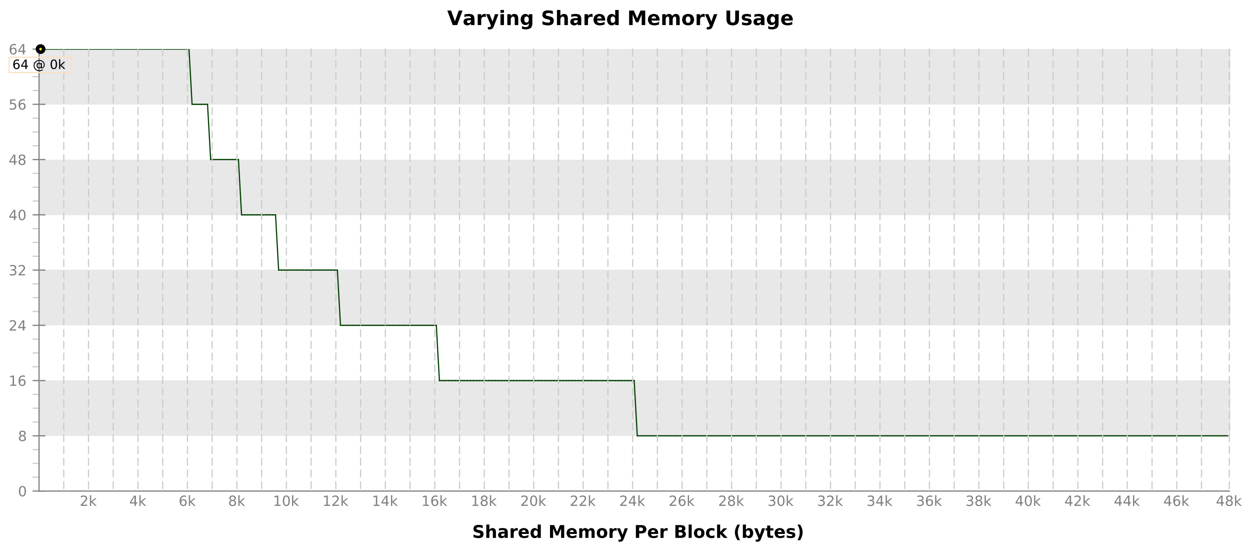
Variable	Achieved	Theoretical	Device Limit	Grid Size: [160,480,1] (76800 blocks) Block Size: [16,16,1] (256
Occupancy Per SM				
Active Blocks		8	16	
Active Warps	50.78	64	64	
Active Threads		2048	2048	
Occupancy	79.3%	100%	100%	
Warps				
Threads/Block		256	1024	
Warps/Block		8	32	
Block Limit		8	16	
Registers				
Registers/Thread		8	255	
Registers/Block		2048	65536	
Block Limit		32	16	
Shared Memory				
Shared Memory/Block		0	49152	
Block Limit			16	

3.3. Occupancy Charts

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




Varying Register Count
64 @ 8
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


Varying Shared Memory Usage
64 @ 0k
64
56
48
40
32
24
16
8
0
2k
4k
6k
8k
10k
12k
14k
16k
18k
20k
22k
24k
26k
28k
30k
32k
34k
36k
38k
40k
42k
44k
46k
48k
Shared Memory Per Block (bytes)

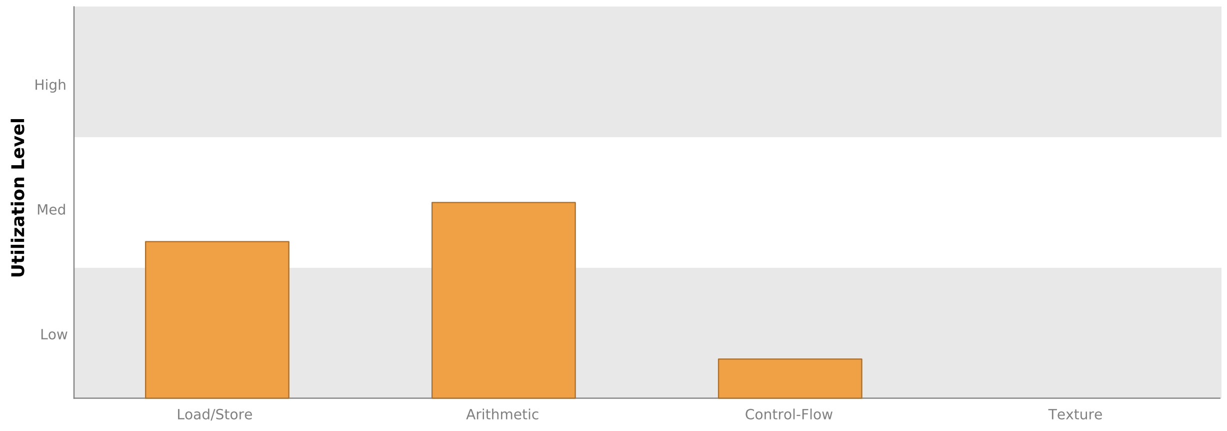
4. Compute Resources

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

4.1. Function Unit Utilization

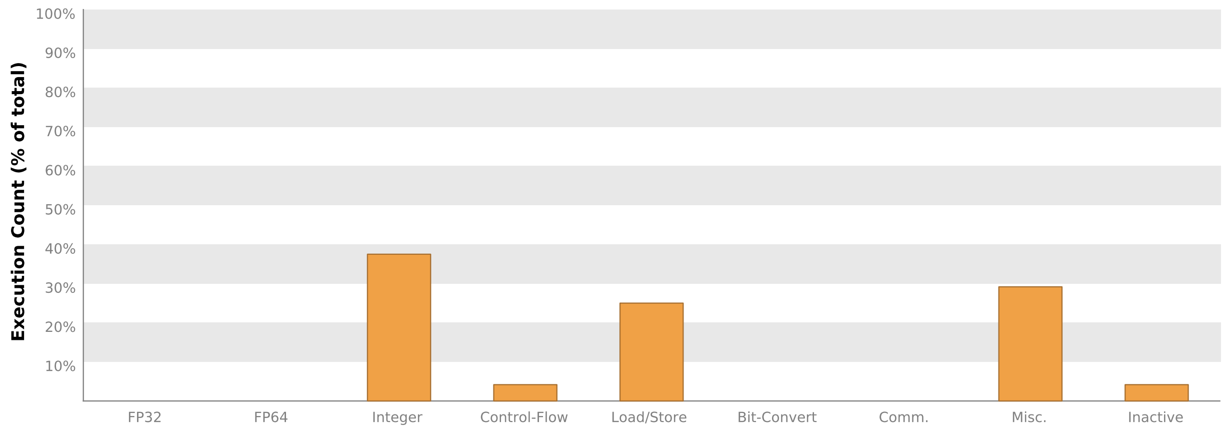
Different types of instructions are executed on different function units within each SM. Performance can be limited if a function unit is over-used by the instructions executed by the kernel. The following results show that the kernel's performance is not limited by overuse of any function unit.

Load/Store - Load and store instructions for local, shared, global, constant, etc. memory.
Arithmetic - All arithmetic instructions including integer and floating-point add and multiply, logical and binary operations, etc.
Control-Flow - Direct and indirect branches, jumps, and calls.
Texture - Texture operations.



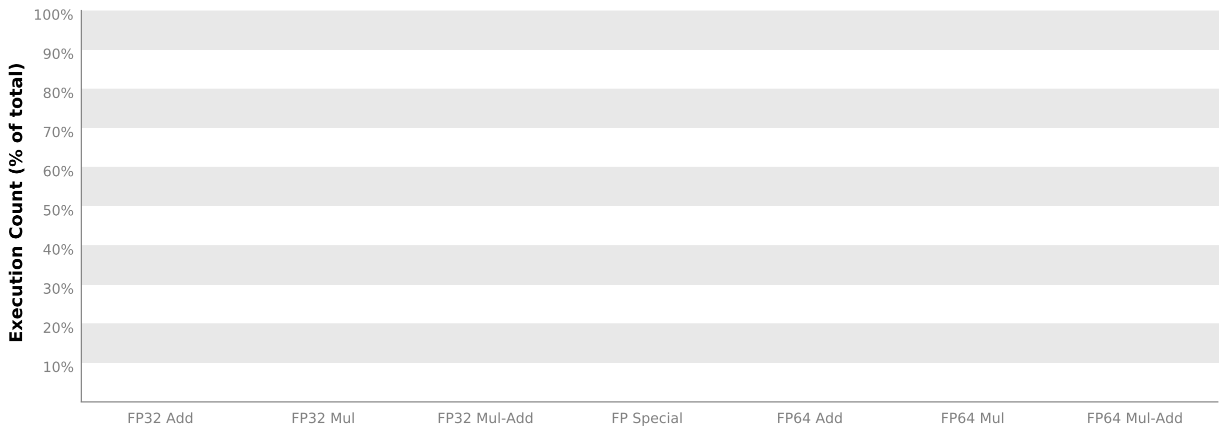
4.2. Instruction Execution Counts

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.


Execution Count (% of total)
FP32 Add
FP32 Mul
FP32 Mul-Add
FP Special
FP64 Add
FP64 Mul
FP64 Mul-Add