Analysis Report

v4

|	
Duration	382.306 μs
Grid Size	[1,1,1]
Block Size	[1024,1,1]
Registers/Thread	30
Shared Memory/Block	28 KiB
Shared Memory Requested	64 KiB
Shared Memory Executed	64 KiB
Shared Memory Bank Size	4 B

[0] Quadro K620

GPU UUID	GPU-1ef8bc5c-b0a8-9167-636d-16994a50ac47
Compute Capability	5.0
Max. Threads per Block	1024
Max. Threads per Multiprocessor	2048
Max. Shared Memory per Block	48 KiB
Max. Shared Memory per Multiprocessor	64 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	863.232 GigaFLOP/s
Double Precision FLOP/s	26.976 GigaFLOP/s
Number of Multiprocessors	3
Multiprocessor Clock Rate	1.124 GHz
Concurrent Kernel	true
Max IPC	6
Threads per Warp	32
Global Memory Bandwidth	28.8 GB/s
Global Memory Size	1.946 GiB
Constant Memory Size	64 KiB
L2 Cache Size	2 MiB
Memcpy Engines	1
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 "v4" is most likely limited by instruction and memory latency. You should first examine the information in the "Instruction And Memory Latency" section to determine how it is limiting performance.

1.1. Kernel Performance Is Bound By Instruction And Memory Latency

This kernel exhibits low compute throughput and memory bandwidth utilization relative to the peak performance of "Quadro K620". These utilization levels indicate that the performance of the kernel is most likely limited by the latency of arithmetic or memory operations. Achieved compute throughput and/or memory bandwidth below 60% of peak typically indicates latency issues.
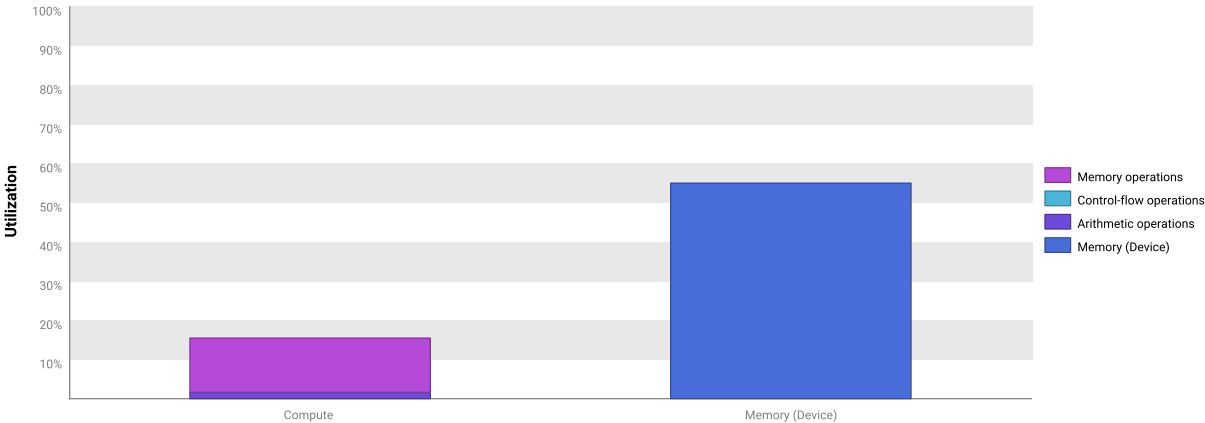
2. Instruction and Memory Latency

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 the kernel does not execute enough blocks.
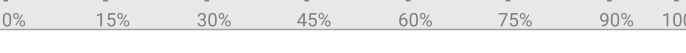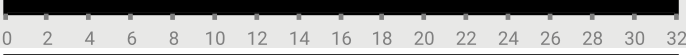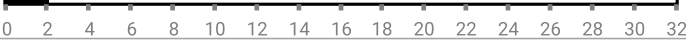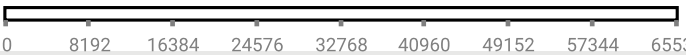
2.1. Grid Size Too Small To Hide Compute And Memory Latency

The kernel does not execute enough blocks to hide memory and operation latency. Typically the kernel grid size must be large enough to fill the GPU with multiple "waves" of blocks. Based on theoretical occupancy, device "Quadro K620" can simultaneously execute 2 blocks on each of the 3 SMs, so the kernel may need to execute a multiple of 6 blocks to hide the compute and memory latency. If the kernel is executing concurrently with other kernels then fewer blocks will be required because the kernel is sharing the SMs with those kernels.

Optimization: Increase the number of blocks executed by the kernel.
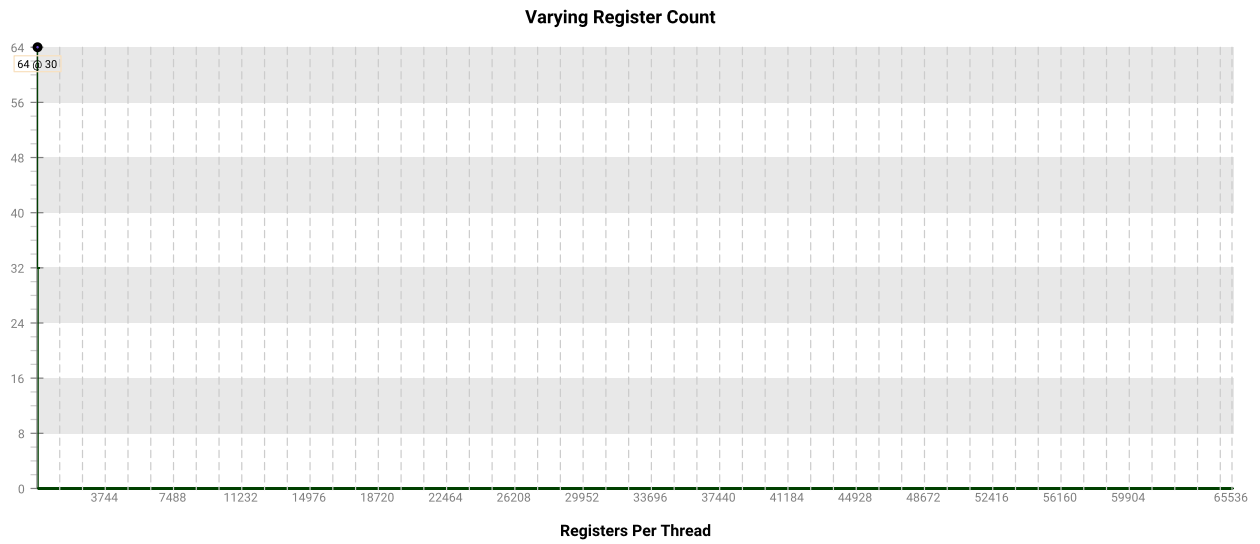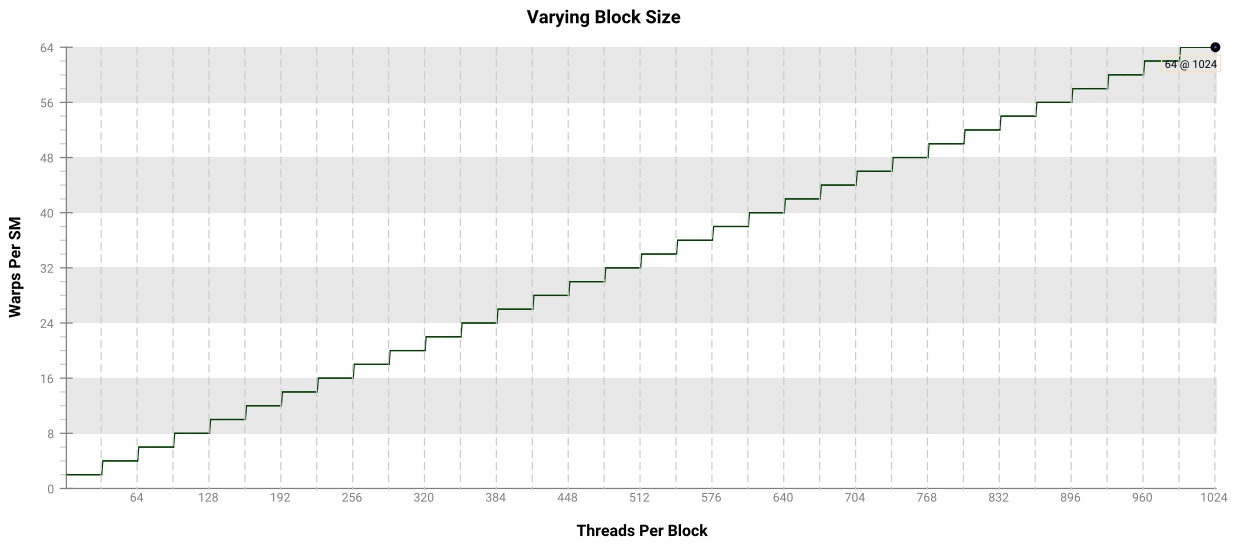
2.2. Occupancy Is Not Limiting Kernel Performance

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: [1,1,1] (1 block) Block Size: [1024,1,1] (1024 threads)
Occupancy Per SM				
Active Blocks		2	32	
Active Warps	32	64	64	
Active Threads		2048	2048	
Occupancy	50%	100%	100%	
Warps				
Threads/Block		1024	1024	
Warps/Block		32	32	
Block Limit		2	32	
Registers				
Registers/Thread		30	65536	
Registers/Block		32768	65536	
Block Limit		2	32	
Shared Memory				
Shared Memory/Block		28672	65536	
Block Limit		2	32	

2.3. Occupancy Charts

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

Varying Block Size

Warps Per SM

64 @ 1024

Threads Per Block

Varying Register Count

64 @ 30

Registers Per Thread

Varying Shared Memory Usage

64 @ 28k

Shared Memory Per Block (bytes)
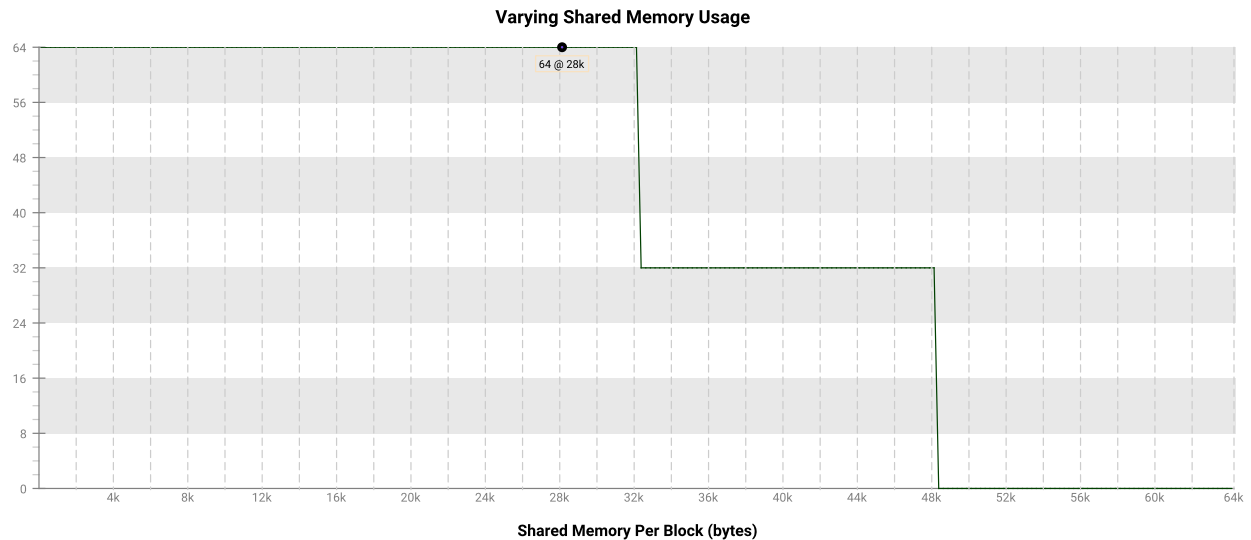
2.4. Multiprocessor Utilization

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.

Utilization

SM 0   SM 1   SM 2

Multiprocessor
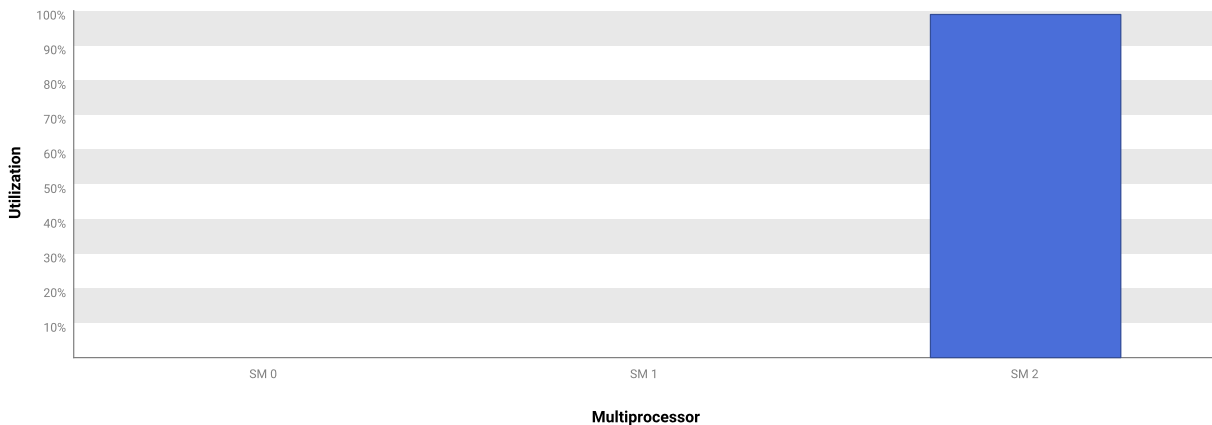
2.5. Instruction Latencies

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 low theoretical or achieved occupancy. Therefore, it is likely that the instruction stall reasons described below are not the primary limiters of performance and so should not be considered until any occupancy issues are resolved.
 Constant - A constant load is blocked due to a miss in the constants cache.
 Instruction Fetch - The next assembly instruction has not yet been fetched.
 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.

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.

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

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.

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

Stall Reasons
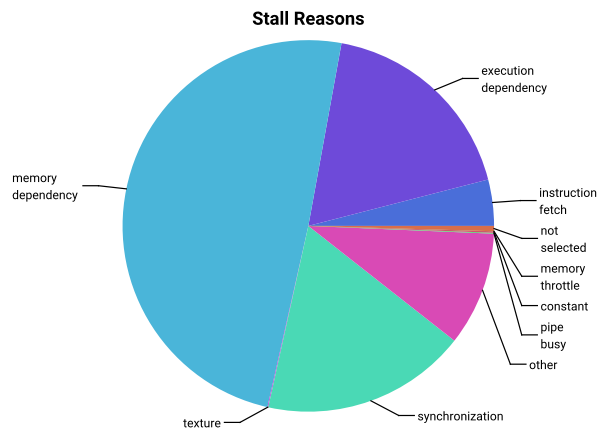
3. Compute Resources

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

## 3.1. 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.
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.
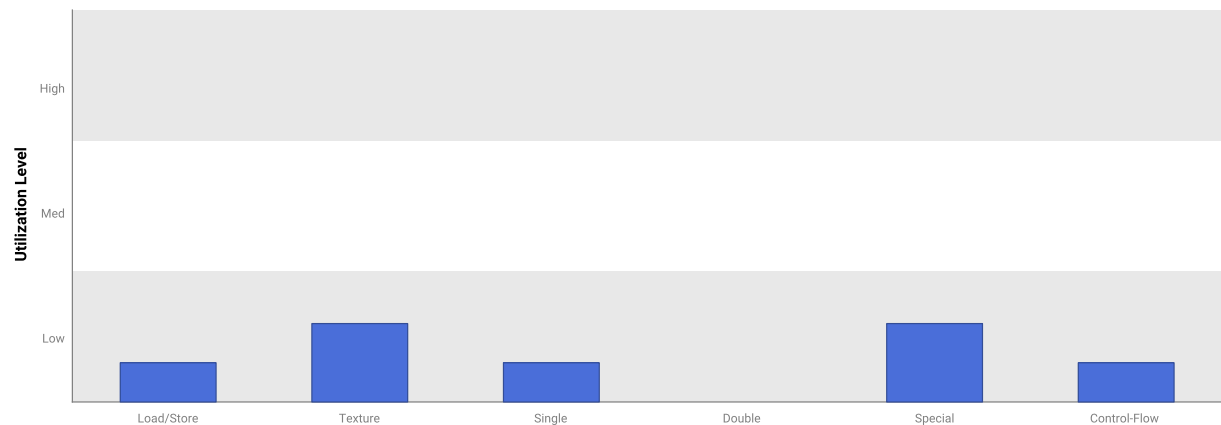
3.2. Instruction Execution Counts

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.
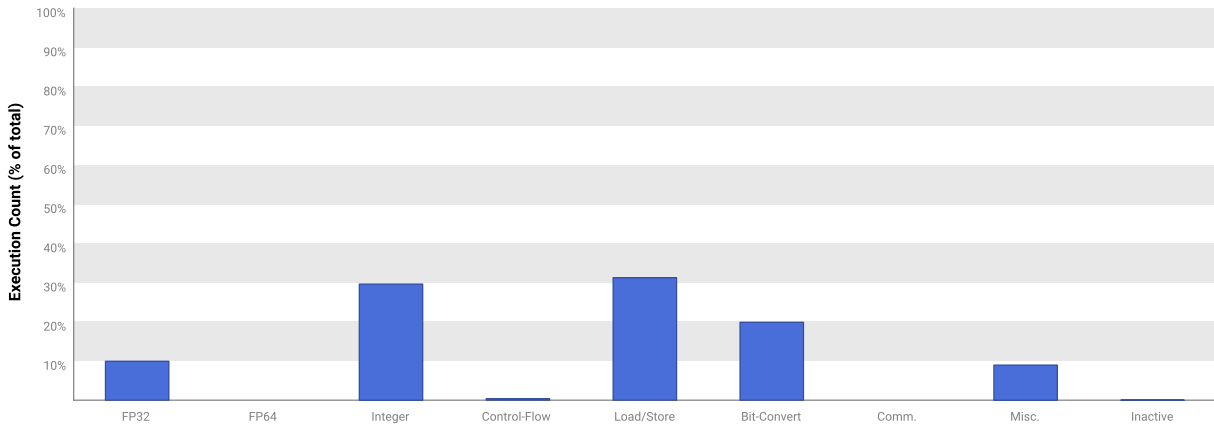
3.3. Floating-Point Operation Counts

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.
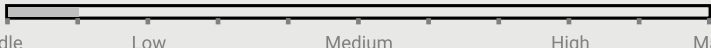
4. Memory Bandwidth

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.

4.1. 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	28672	10.886 GB/s	
Shared Stores	28672	10.886 GB/s	
Shared Total	57344	21.772 GB/s	Idle　Low　Medium　High　Max
L2 Cache			
Reads	119292	11.323 GB/s	
Writes	16390	1.556 GB/s	
Total	135682	12.878 GB/s	Idle　Low　Medium　High　Max
Unified Cache			
Local Loads	0	0 B/s	
Local Stores	0	0 B/s	
Global Loads	229376	10.886 GB/s	
Global Stores	16384	1.555 GB/s	
Texture Reads	114688	10.886 GB/s	
Unified Total	360448	23.327 GB/s	Idle　Low　Medium　High　Max
Device Memory			
Reads	121936	11.574 GB/s	
Writes	36555	3.47 GB/s	
Total	158491	15.043 GB/s	Idle　Low　Medium　High　Max
System Memory			
[PCIe configuration: Gen2 x16, 5 Gbit/s]			
Reads	0	0 B/s	Idle　Low　Medium　High　Max
Writes	5	474.581 kB/s	Idle　Low　Medium　High　Max