

Analysis Report

quadratic_difference

Duration	121.981 ms (121,980,577 ns)
Grid Size	[700000,47,1]
Block Size	[2,32,1]
Registers/Thread	23
Shared Memory/Block	560 B
Shared Memory Requested	96 KiB
Shared Memory Executed	96 KiB
Shared Memory Bank Size	4 B

[0] GeForce GTX TITAN X

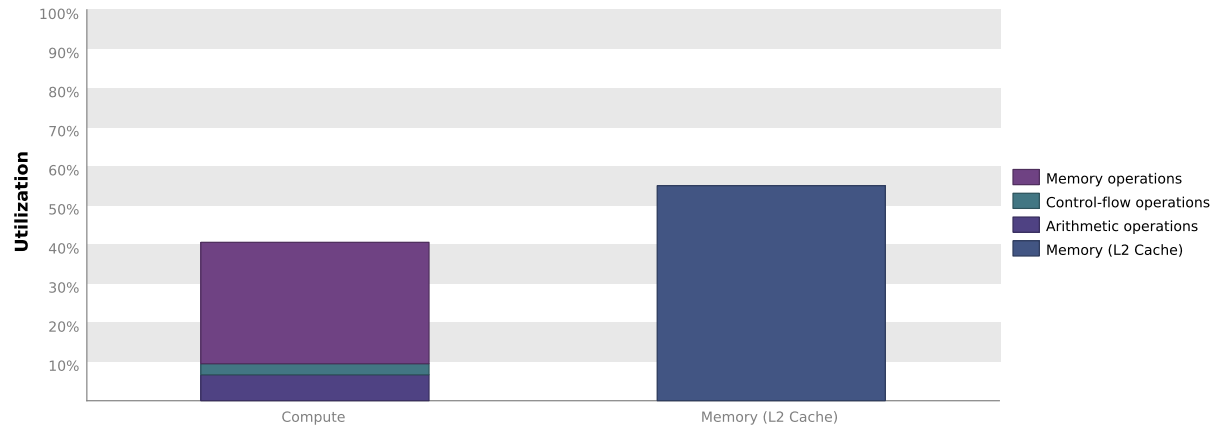
GPU UUID	GPU-0a4eae89-2e08-4851-bc8f-213711d16d38
Compute Capability	5.2
Max. Threads per Block	1024
Max. Shared Memory per Block	48 KiB
Max. Registers per Block	65536
Max. Grid Dimensions	[2147483647, 65535, 65535]
Max. Block Dimensions	[1024, 1024, 64]
Max. Warps per Multiprocessor	64
Max. Blocks per Multiprocessor	32
Single Precision FLOP/s	6.611 TeraFLOP/s
Double Precision FLOP/s	206.592 GigaFLOP/s
Number of Multiprocessors	24
Multiprocessor Clock Rate	1.076 GHz
Concurrent Kernel	true
Max IPC	6
Threads per Warp	32
Global Memory Bandwidth	336.48 GB/s
Global Memory Size	11.999 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 "quadratic_difference" 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 "GeForce GTX TITAN X". 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 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.

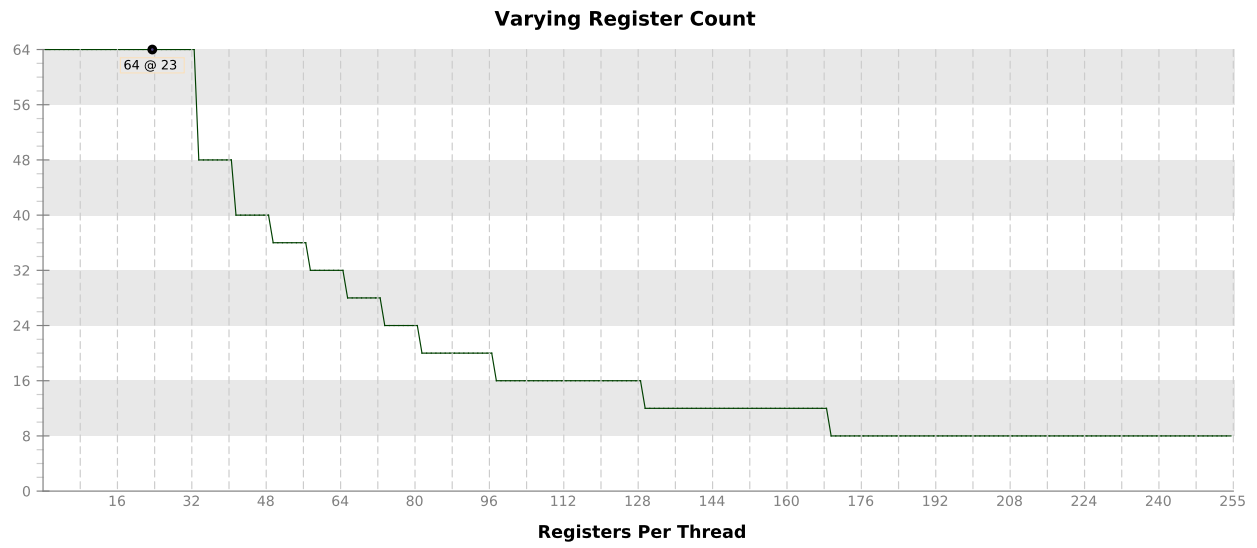
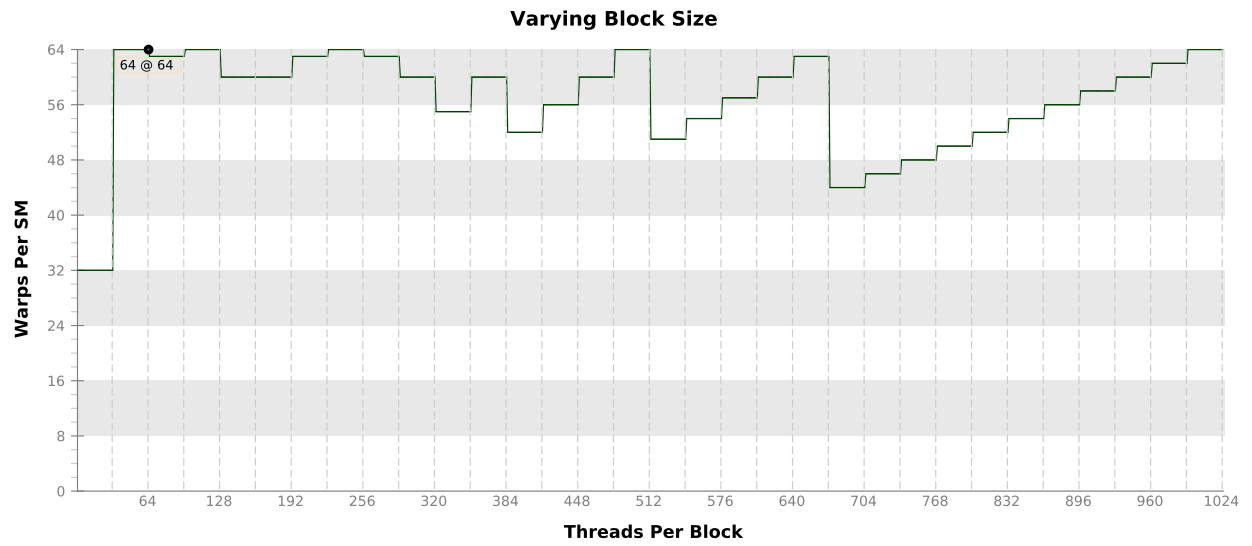
2.1. Occupancy Is Not Limiting Kernel Performance

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

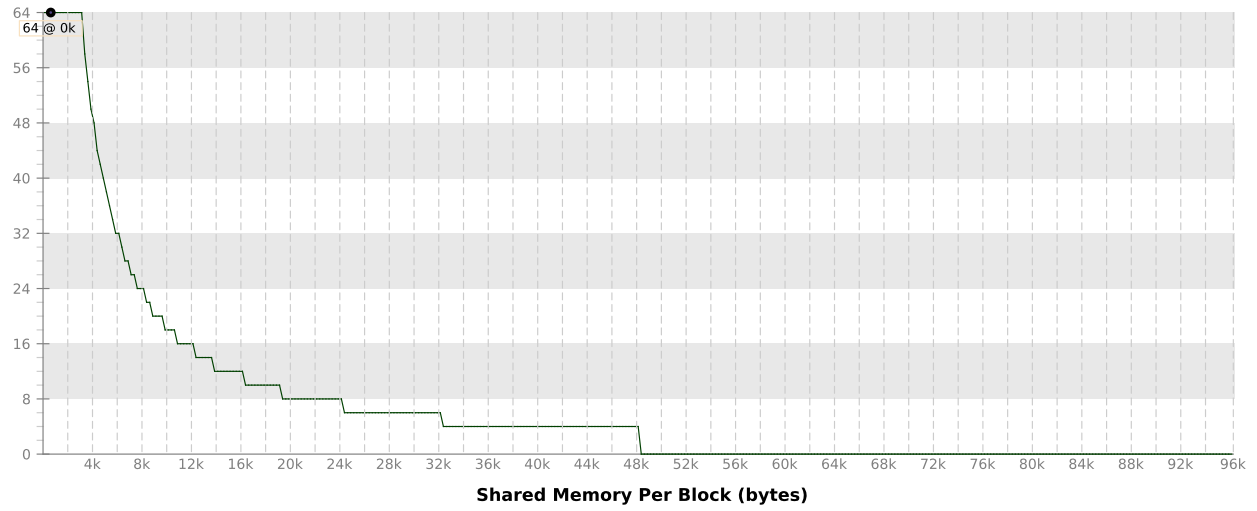
Variable	Achieved	Theoretical	Device Limit	Grid Size: [700000,47,1] (32900000 blocks) Block Size
Occupancy Per SM				
Active Blocks		32	32	
Active Warps	58.34	64	64	
Active Threads		2048	2048	
Occupancy	91.2%	100%	100%	
Warps				
Threads/Block		64	1024	
Warps/Block		2	32	
Block Limit		32	32	
Registers				
Registers/Thread		23	255	
Registers/Block		1536	65536	
Block Limit		42	32	
Shared Memory				
Shared Memory/Block		560	98304	
Block Limit		128	32	

2.2. Occupancy Charts

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



Varying Shared Memory Usage



3. 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 all threads in a warp have the same branching and predication behavior. The results below indicate that a significant fraction of the available compute performance is being wasted because branch and predication behavior is differing for threads within a warp.

3.1. Low Warp Execution Efficiency

Warp execution efficiency is the average percentage of active threads in each executed warp. Increasing warp execution efficiency will increase utilization of the GPU's compute resources. The kernel's warp execution efficiency of 77.3% is less than 100% due to divergent branches and predicated instructions. If predicated instructions are not taken into account the warp execution efficiency for these kernels is 89.3%.

Optimization: Reduce the amount of intra-warp divergence and predication in the kernel.

3.2. Divergent Branches

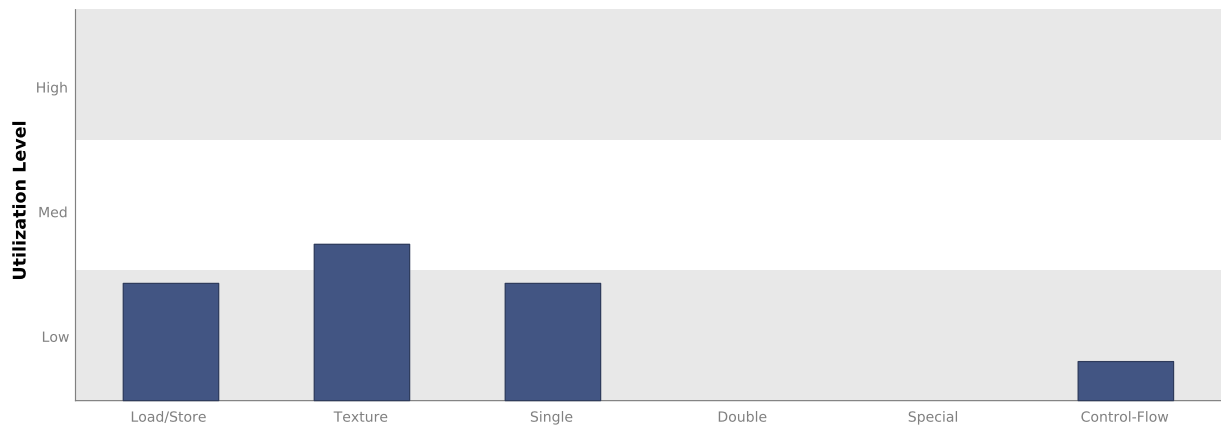
Compute resource are used most efficiently when all threads in a warp have the same branching behavior. When this does not occur the branch is said to be divergent. Divergent branches lower warp execution efficiency which leads to inefficient use of the GPU's compute resources.

Optimization: Each entry below points to a divergent branch within the kernel. For each branch reduce the amount of intra-warp divergence.

3.3. 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.

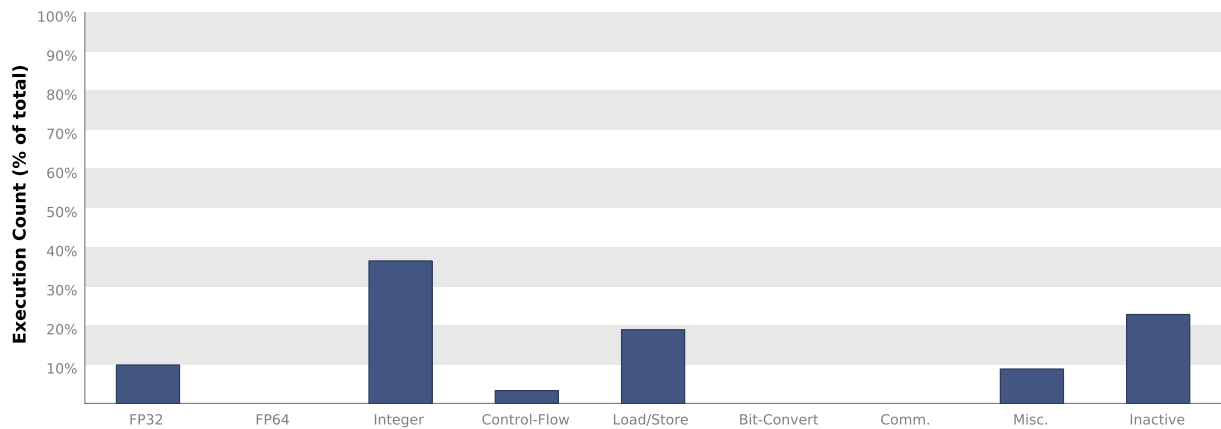
- Load/Store - Load and store instructions for shared and constant memory.
- Texture - Load and store instructions for local, global, and texture memory.
- Single - Single-precision integer and floating-point arithmetic instructions.
- Double - Double-precision floating-point arithmetic instructions.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



3.4. Instruction Execution Counts

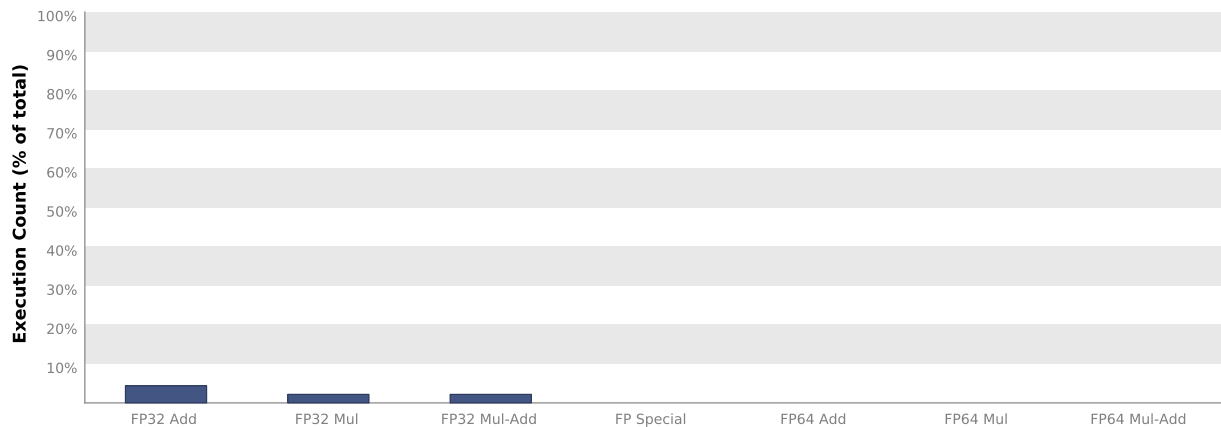
The following chart shows the mix of instructions executed by the kernel. The instructions are grouped into classes and for each

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.



3.5. 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 L2 cache.






4.1. 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.

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.

4.2. 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	526120256	574.19 GB/s	
Shared Stores	657650696	717.738 GB/s	
Shared Total	1183770952	1,291.928 GB/s	
L2 Cache			
Reads	1641415859	447.847 GB/s	
Writes	235984566	64.386 GB/s	
Total	1877400425	512.233 GB/s	
Unified Cache			
Local Loads	0	0 B/s	
Local Stores	0	0 B/s	
Global Loads	3545698192	447.821 GB/s	
Global Stores	235984560	64.386 GB/s	
Texture Reads	2236002816	610.075 GB/s	
Unified Total	6017685568	1,122.282 GB/s	
Device Memory			
Reads	32985619	9 GB/s	
Writes	123231000	33.623 GB/s	
Total	156216619	42.622 GB/s	
System Memory			
[PCIe configuration: Gen3 x16, 8 Gbit/s]			
Reads	0	0 B/s	
Writes	5	1.364 kB/s	