Analysis Report

horizontal_diffusion_gpu(int, int, int, int, int, int, int, int, int, int, int, int, int, int, int, int, int, int, int, int, int, int, int, int, int, int, char, float*, float*, float*, float*, float*, float*, float*, float*, float*, float*, float, float*, float, float)

Duration	15.084 ms (15,084,083 ns)
Grid Size	[55,39,1]
Block Size	[8,8,1]
Registers/Thread	63
Shared Memory/Block	1.953 KiB
Shared Memory Requested	16 KiB
Shared Memory Executed	16 KiB
Shared Memory Bank Size	4 B

[0] GeForce GT 730

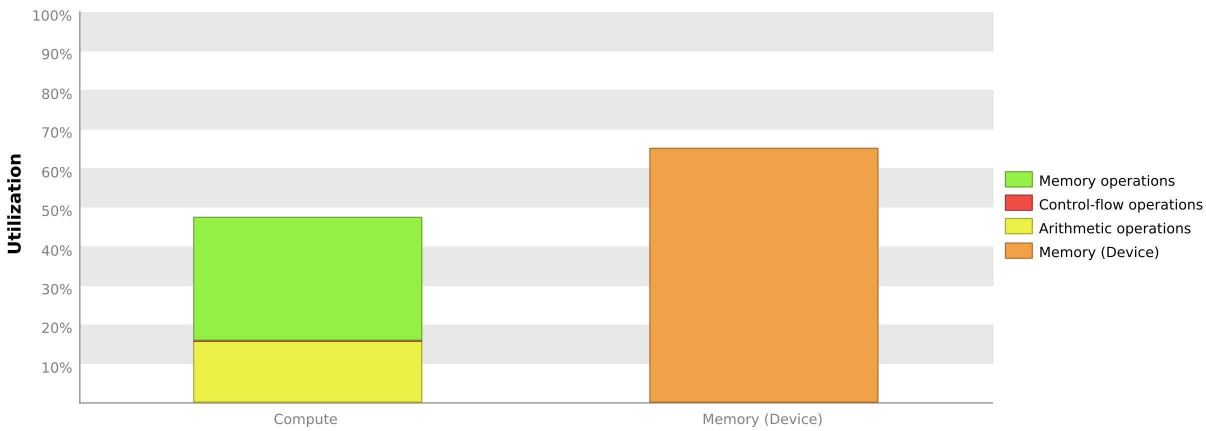
Compute Capability	2.1
Max. Threads per Block	1024
Max. Shared Memory per Block	48 KiB
Max. Registers per Block	32768
Max. Grid Dimensions	[65535, 65535, 65535]
Max. Block Dimensions	[1024, 1024, 64]
Max. Warps per Multiprocessor	48
Max. Blocks per Multiprocessor	8
Number of Multiprocessors	2
Multiprocessor Clock Rate	1.4 GHz
Concurrent Kernel	true
Max IPC	4
Threads per Warp	32
Global Memory Bandwidth	22.4 GB/s
Global Memory Size	2 GiB
Constant Memory Size	64 KiB
L2 Cache Size	128 KiB
Memcpy Engines	1
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 "horizontal_diffusion_gpu" 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 730" 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. 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	126848	1.068 GB/s	
Shared Stores	158560	1.335 GB/s	
Global Loads	8349080	70.272 GB/s	
Global Stores	661090	2.127 GB/s	
Atomic	0	0 B/s	
L1/Shared Total	9295578	74.802 GB/s	Idle Low Medium High Max
L2 Cache			
L1 Reads	9045456	19.033 GB/s	
L1 Writes	1011076	2.127 GB/s	
Texture Reads	0	0 B/s	
Atomic	0	0 B/s	
Total	10056532	21.161 GB/s	Idle Low Medium High Max
Texture Cache			
Reads	0	0 B/s	Idle Low Medium High Max
Device Memory			
Reads	5774649	12.151 GB/s	
Writes	817158	1.719 GB/s	
Total	6591807	13.87 GB/s	Idle Low Medium High Max
System Memory			
[PCIe configuration: Gen2 x4, 5 Gbit/s]			
Reads	0	0 B/s	Idle Low Medium High Max
Writes	0	0 B/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 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 number of registers used by the kernel.

3.1. GPU Utilization Is Limited By Register Usage

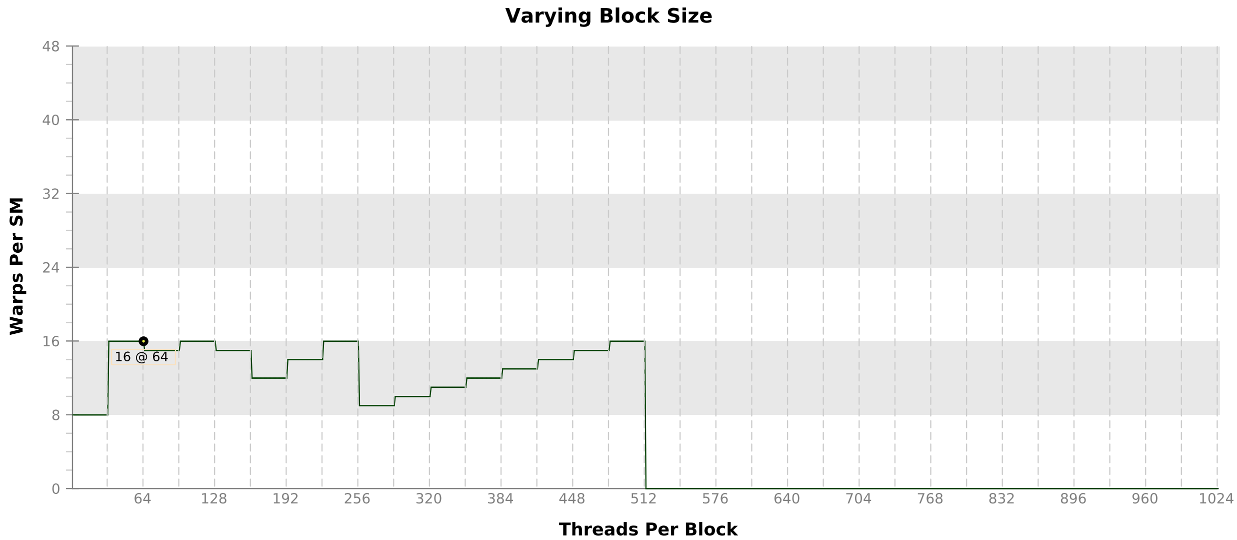
The kernel uses 63 registers for each thread (4032 registers for each block). This register usage is likely preventing the kernel from fully utilizing the GPU. Device "GeForce GT 730" provides up to 32768 registers for each block. Because the kernel uses 4032 registers for each block each SM is limited to simultaneously executing 8 blocks (16 warps). Chart "Varying Register Count" below shows how changing register usage will change the number of blocks that can execute on each SM.

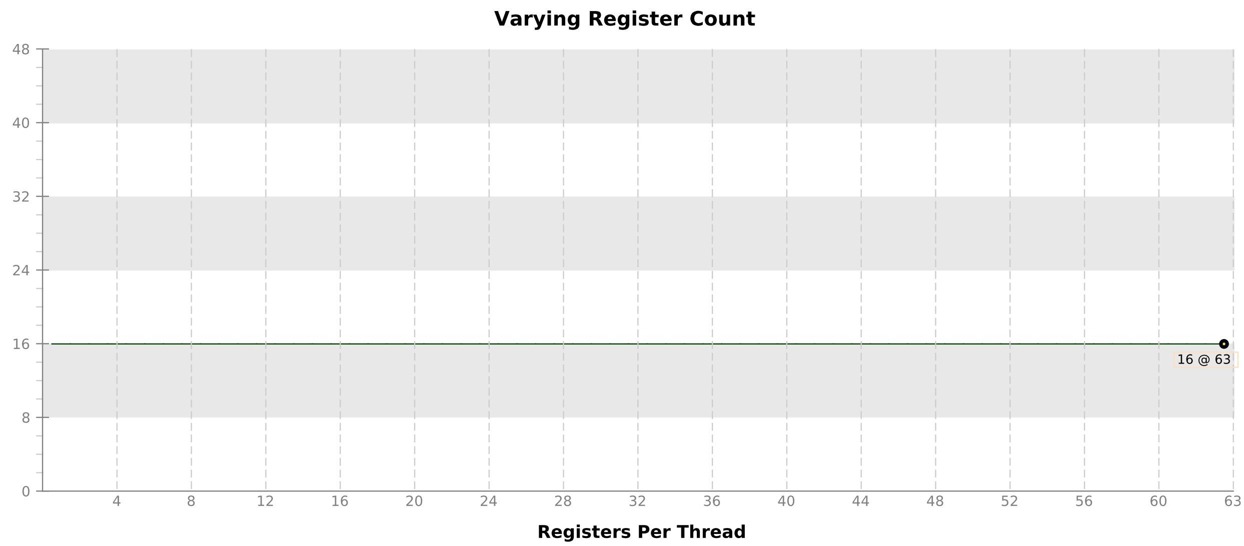
Optimization: Use the -maxrregcount flag or the __launch_bounds__ qualifier to decrease the number of registers used by each thread. This will increase the number of blocks that can execute on each SM. On devices with Compute Capability 5.2 turning global cache off can increase the occupancy limited by register usage.

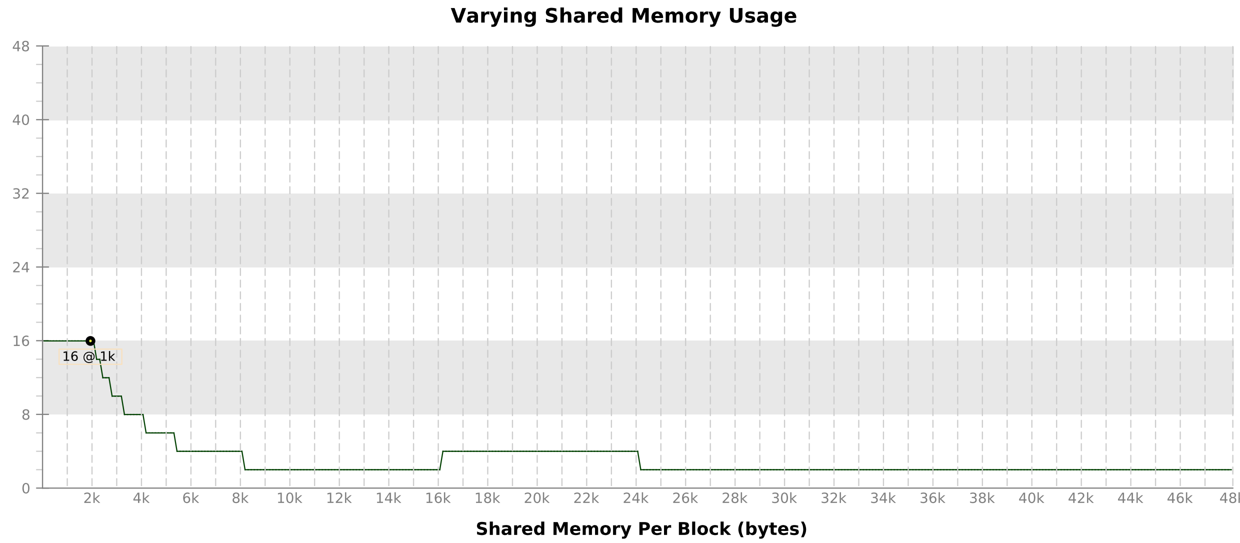
Variable	Achieved	Theoretical	Device Limit	Grid Size: [55,39,1] (2145 blocks) Block Size: [8,8,1] (64 threads
Occupancy Per SM				
Active Blocks		8	8	
Active Warps	15.69	16	48	
Active Threads		512	1536	
Occupancy	32.7%	33.3%	100%	
Warps				
Threads/Block		64	1024	
Warps/Block		2	32	
Block Limit		24	8	
Registers				
Registers/Thread		63	63	
Registers/Block		4096	32768	
Block Limit		8	8	
Shared Memory				
Shared Memory/Block		2000	16384	
Block Limit		8	8	

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


Varying Register Count
48
40
32
24
16
8
0
16 @ 63
4
8
12
16
20
24
28
32
36
40
44
48
52
56
60
63
Registers Per Thread


Varying Shared Memory Usage
48
40
32
24
16
8
0
16 @ 1k
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)

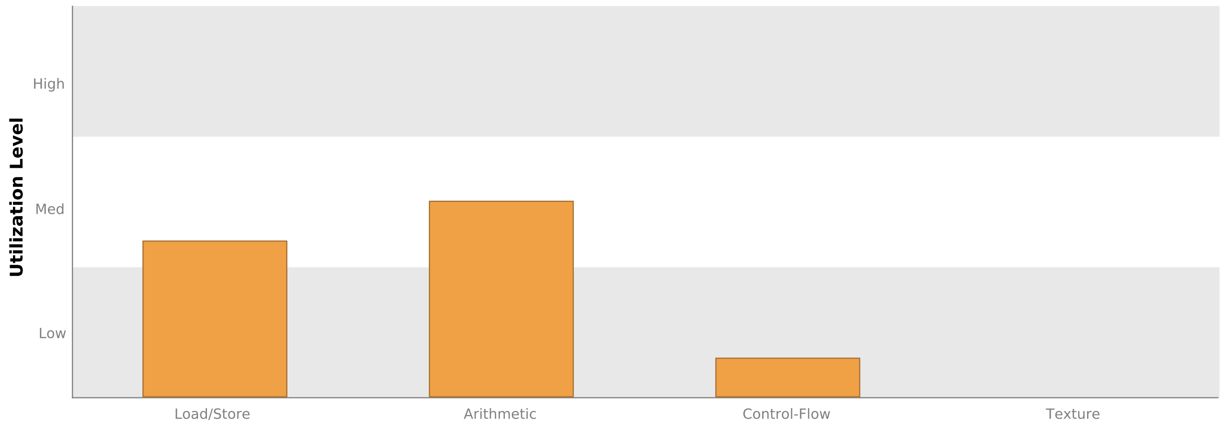
4. Compute Resources

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

4.1. Function Unit Utilization

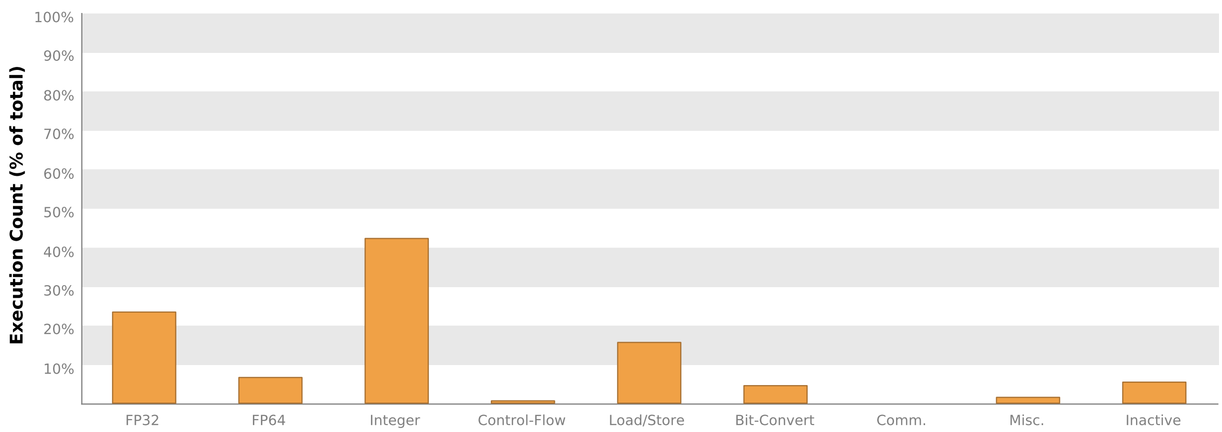
Different types of instructions are executed on different function units within each SM. Performance can be limited if a function unit is over-used by the instructions executed by the kernel. The following results show that the kernel's performance is not limited by overuse of any function unit.

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.