Analysis Report

compute_util_table_ver_1(int*, int**, unsigned int, unsigned int, int, int, int, int, int, int, int, bool)

Duration	206.929 ms (206,929,295 ns)
Grid Size	[381470,1,1]
Block Size	[128,1,1]
Registers/Thread	33
Shared Memory/Block	2.543 KiB
Shared Memory Requested	48 KiB
Shared Memory Executed	48 KiB
Shared Memory Bank Size	4 B

[0] GeForce GTX TITAN

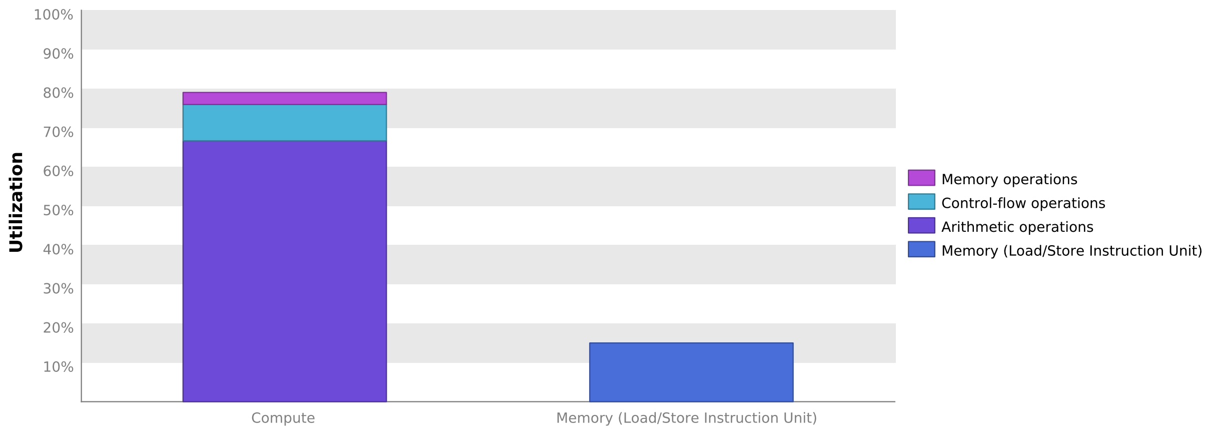
GPU UUID	GPU-c2e0ee17-52b9-35a2-645d-ea8f92144945
Compute Capability	3.5
Max. Threads per Block	1024
Max. Shared Memory per Block	48 KiB
Max. Registers per Block	65536
Max. Grid Dimensions	[2147483647, 65535, 65535]
Max. Block Dimensions	[1024, 1024, 64]
Max. Warps per Multiprocessor	64
Max. Blocks per Multiprocessor	16
Single Precision FLOP/s	4.707 TeraFLOP/s
Double Precision FLOP/s	196.112 GigaFLOP/s
Number of Multiprocessors	14
Multiprocessor Clock Rate	875.5 MHz
Concurrent Kernel	true
Max IPC	7
Threads per Warp	32
Global Memory Bandwidth	288.384 GB/s
Global Memory Size	5.999 GiB
Constant Memory Size	64 KiB
L2 Cache Size	1.5 MiB
Memcpy Engines	1
PCIe Generation	2
PCIe Link Rate	5 Gbit/s
PCIe Link Width	8

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 "compute_util_table_ver_1" is most likely limited by compute. You should first examine the information in the "Compute Resources" section to determine how it is limiting performance.

1.1. Kernel Performance Is Bound By Compute

For device "GeForce GTX TITAN" the kernel's memory utilization is significantly lower than its compute utilization. These utilization levels indicate that the performance of the kernel is most likely being limited by computation on the SMs.



2. 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.

2.1. 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.

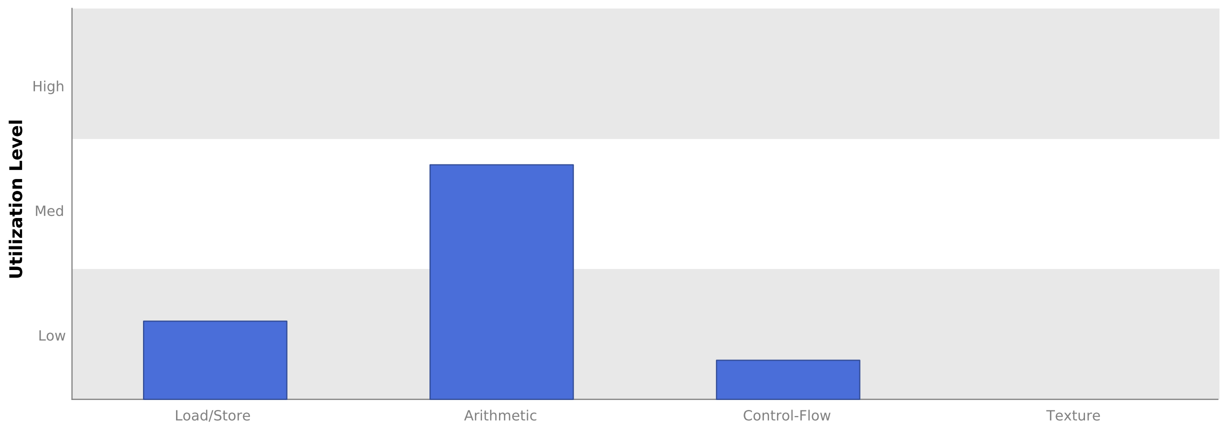
Optimization: Each entry below points to a divergent branch within the kernel. For each branch reduce the amount of intra-warp divergence.

/home/grad12/ffiorett/git/CUDA/cudaDBE/Debug/../src/GPU/gpu_dpop_util_phase.cu

Line 555	Divergence = 25% [381470 divergent executions out of 1525879 total executions]

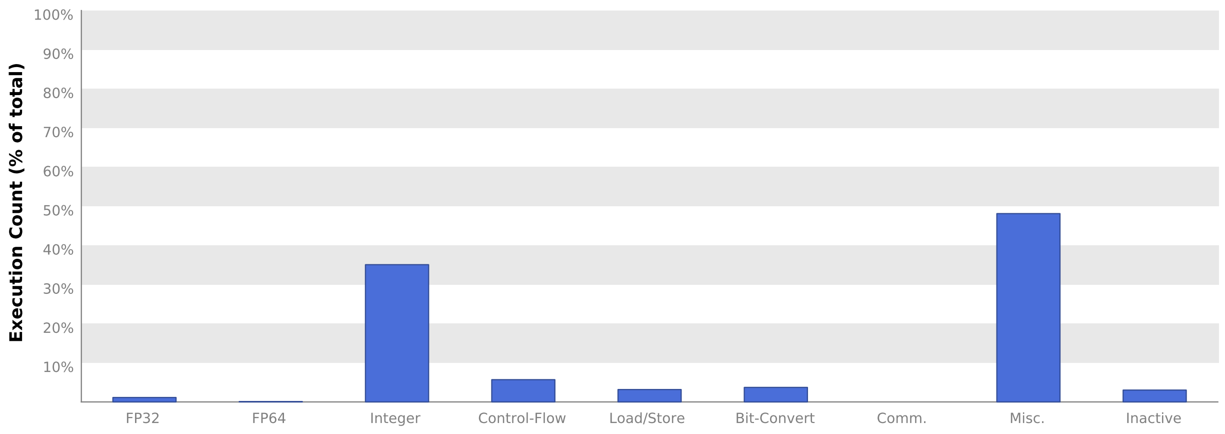
2.2. Function Unit Utilization

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.



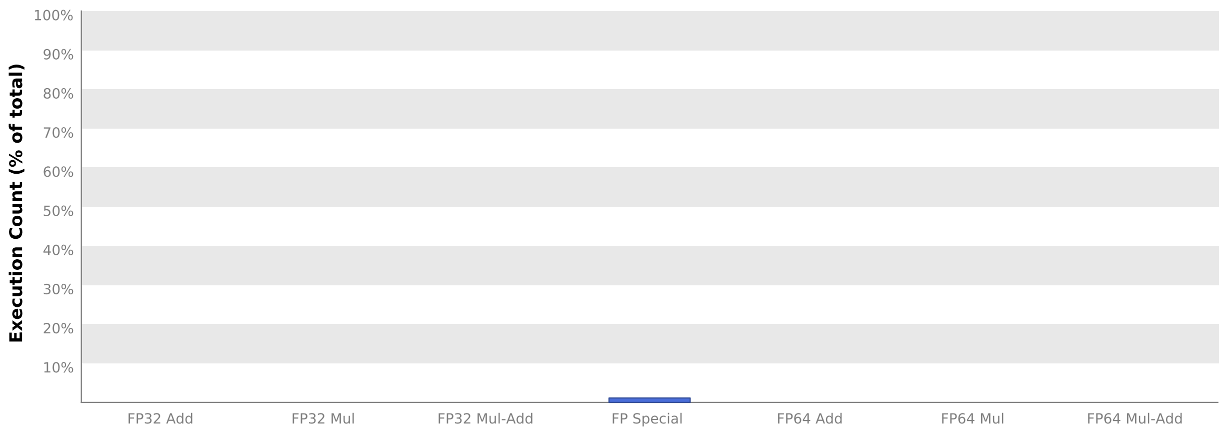
2.3. Instruction Execution Counts

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.



2.4. 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.

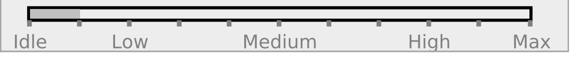
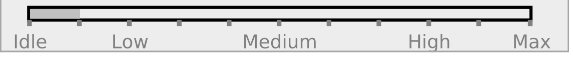
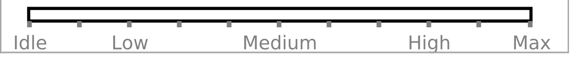
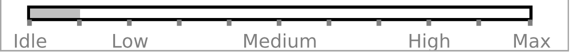
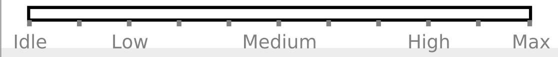
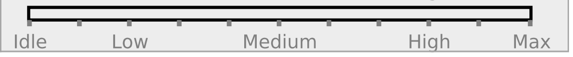


3. Memory Bandwidth

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.

3.1. 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	
L1/Shared Memory			
Local Loads	0	0 B/s	
Local Stores	1525880	1.077 GB/s	
Shared Loads	109100351	154.038 GB/s	
Shared Stores	57601935	81.328 GB/s	
Global Loads	115912848	24.739 GB/s	
Global Stores	1525879	1.077 GB/s	
Atomic	0	0 B/s	
L1/Shared Total	285666893	262.259 GB/s	
L2 Cache			
L1 Reads	140177066	24.739 GB/s	
L1 Writes	6103516	1.077 GB/s	
Texture Reads	0	0 B/s	
Noncoherent Reads	0	0 B/s	
Atomic	0	0 B/s	
Total	146280582	25.817 GB/s	
Texture Cache			
Reads	0	0 B/s	
Device Memory			
Reads	6539578	1.154 GB/s	
Writes	6103598	1.077 GB/s	
Total	12643176	2.231 GB/s	
System Memory			
[PCIe configuration: Gen2 x8, 5 Gbit/s]			
Reads	0	0 B/s	
Writes	0	0 B/s	

4. Instruction and Memory Latency

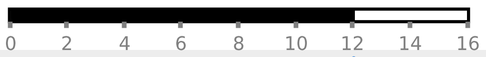
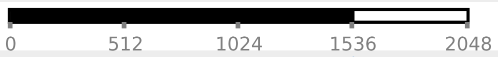
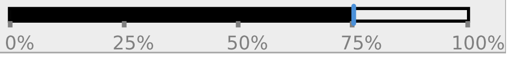
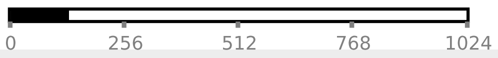
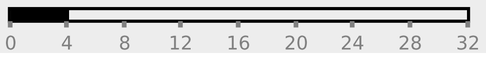
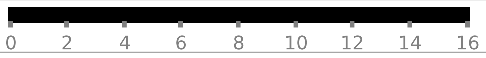
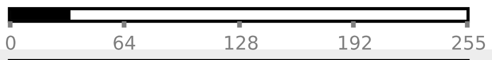
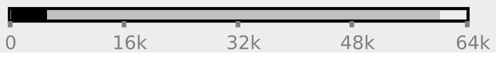
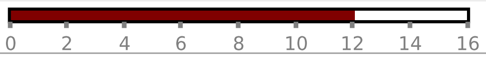
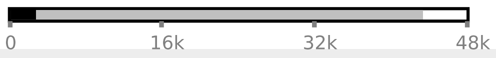
Instruction and memory latency limit the performance of a kernel when the GPU does not have enough work to keep busy. The performance of latency-limited kernels can often be improved by increasing occupancy. Occupancy is a measure of how many warps the kernel has active on the GPU, relative to the maximum 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.

4.1. GPU Utilization May Be Limited By Register Usage

Theoretical occupancy is less than 100% but is large enough that increasing occupancy may not improve performance. You can attempt the following optimization to increase the number of warps on each SM but it may not lead to increased performance.

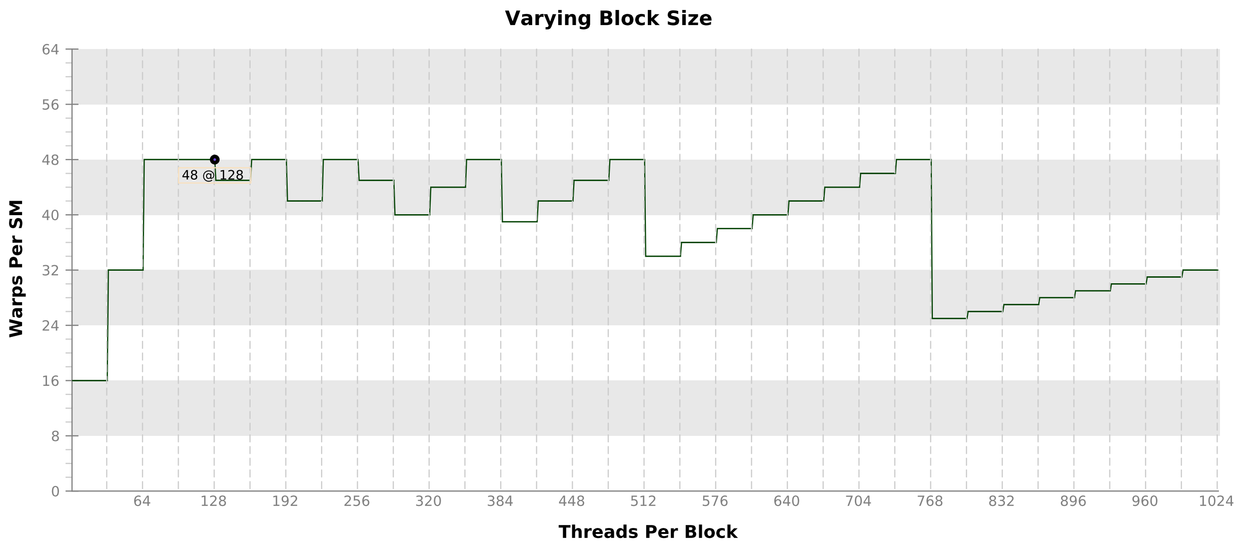
The kernel uses 33 registers for each thread (4224 registers for each block). This register usage is likely preventing the kernel from fully utilizing the GPU. Device "GeForce GTX TITAN" provides up to 65536 registers for each block. Because the kernel uses 4224 registers for each block each SM is limited to simultaneously executing 12 blocks (48 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.

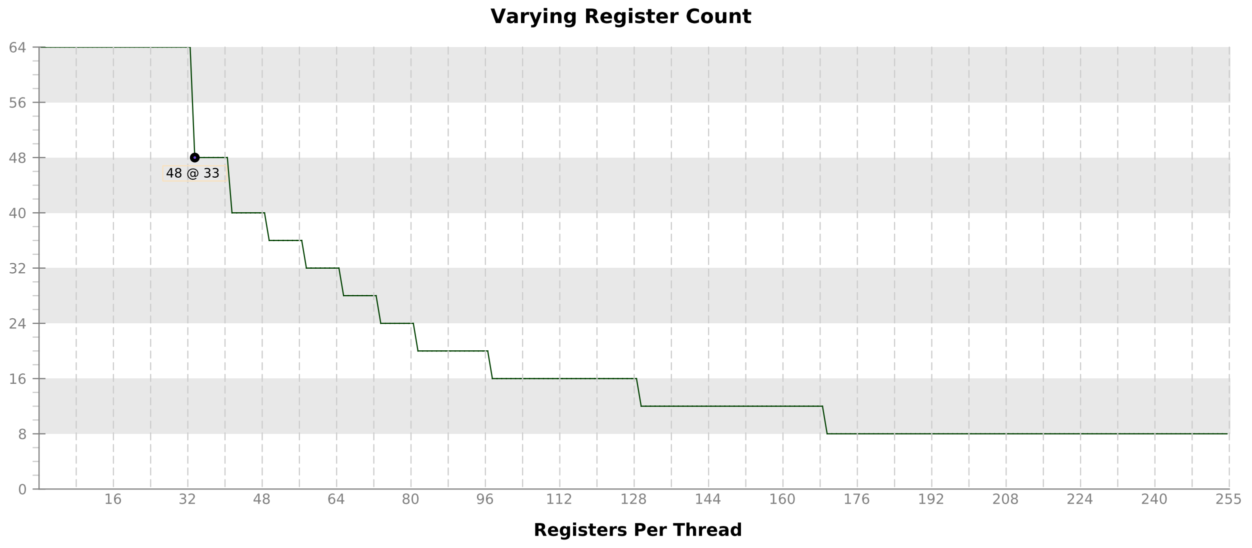
Variable	Achieved	Theoretical	Device Limit	Grid Size: [381470,1,1] (381470 blocks) Block Size: [1
Occupancy Per SM				
Active Blocks		12	16	
Active Warps	47.81	48	64	
Active Threads		1536	2048	
Occupancy	74.7%	75%	100%	
Warps				
Threads/Block		128	1024	
Warps/Block		4	32	
Block Limit		16	16	
Registers				
Registers/Thread		33	255	
Registers/Block		5120	65536	
Block Limit		12	16	
Shared Memory				
Shared Memory/Block		2604	49152	
Block Limit		17	16	

4.2. Occupancy Charts

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

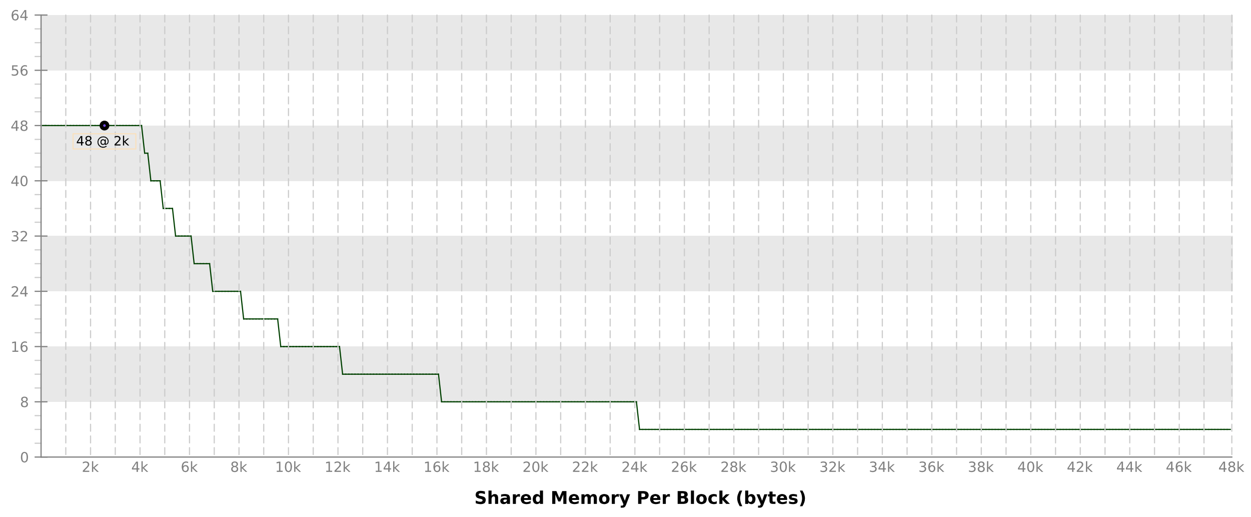


Varying Block Size
Warps Per SM
Threads Per Block
48 @ 128
0
8
16
24
32
40
48
56
64
64
128
192
256
320
384
448
512
576
640
704
768
832
896
960
1024



Varying Register Count
Registers Per Thread
48 @ 33
0
8
16
24
32
40
48
56
64
16
32
48
64
80
96
112
128
144
160
176
192
208
224
240
255

Varying Shared Memory Usage


48 @ 2k
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)