Analysis Report

flock_kernal(float*, float*, float*, float*, float*, float*, float*, float*, float*, float*, int)

Duration	647.382 µs
Grid Size	[32,1,1]
Block Size	[32,32,1]
Registers/Thread	46
Shared Memory/Block	19.555 KiB
Shared Memory Requested	96 KiB
Shared Memory Executed	96 KiB
Shared Memory Bank Size	4 B

[0] Quadro M4000

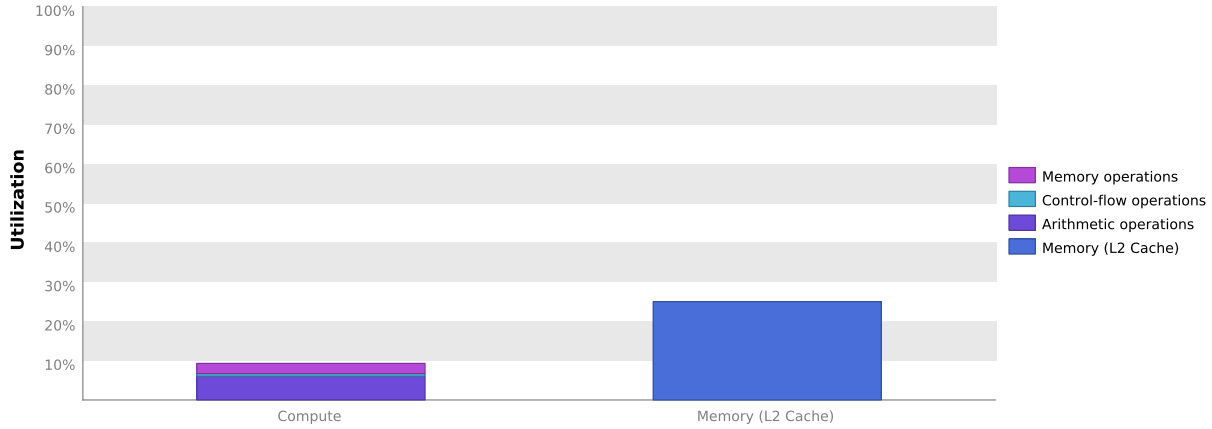
GPU UUID	GPU-45e1bfa2-9dfe-ee51-9716-d030e4dc476d
Compute Capability	5.2
Max. Threads per Block	1024
Max. Shared Memory per Block	48 KiB
Max. Registers per Block	65536
Max. Grid Dimensions	[2147483647, 65535, 65535]
Max. Block Dimensions	[1024, 1024, 64]
Max. Warps per Multiprocessor	64
Max. Blocks per Multiprocessor	32
Single Precision FLOP/s	2.571 TeraFLOP/s
Double Precision FLOP/s	80.34 GigaFLOP/s
Number of Multiprocessors	13
Multiprocessor Clock Rate	772.5 MHz
Concurrent Kernel	true
Max IPC	6
Threads per Warp	32
Global Memory Bandwidth	192.32 GB/s
Global Memory Size	7.931 GiB
Constant Memory Size	64 KiB
L2 Cache Size	2 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 "flock_kernal" 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 M4000". 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. The results below indicate that occupancy can be improved by reducing the number of registers used by the kernel.

2.1. 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.

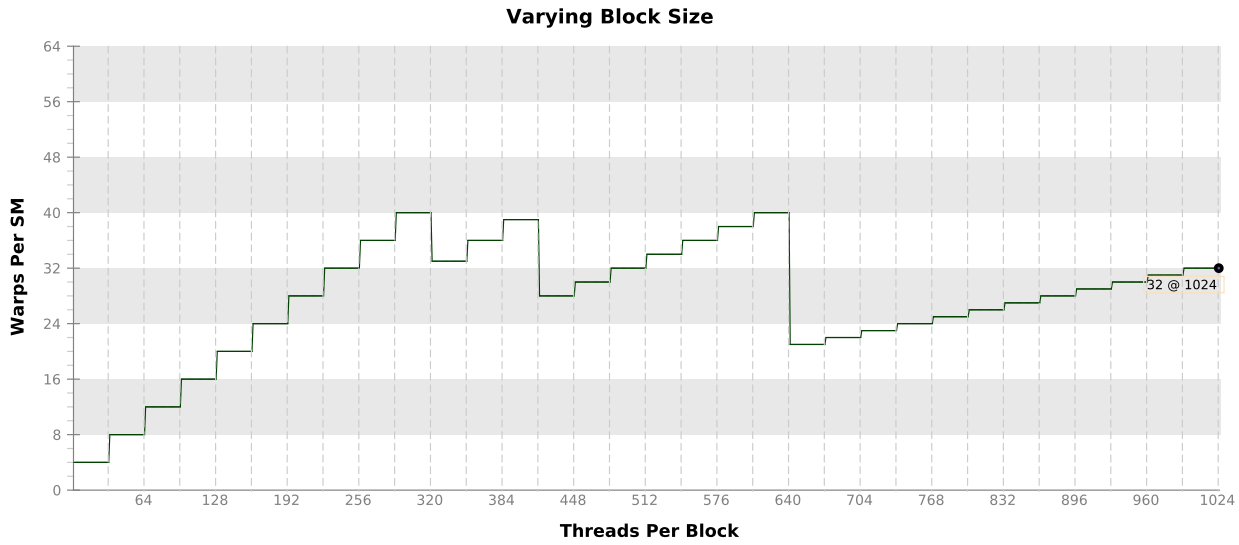
The kernel uses 46 registers for each thread (47104 registers for each block). This register usage is likely preventing the kernel from fully utilizing the GPU. Device "Quadro M4000" provides up to 65536 registers for each block. Because the kernel uses 47104 registers for each block each SM is limited to simultaneously executing 1 block (32 warps). Chart "Varying Register Count" below shows how changing register usage will change the number of blocks that can execute on each SM.

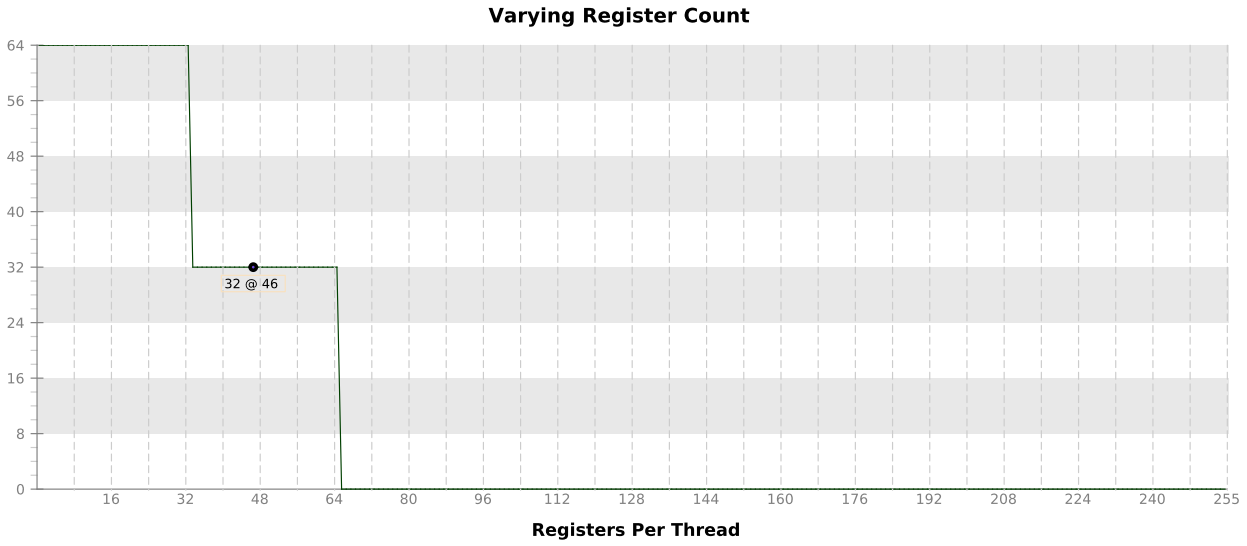
Optimization: Use the -maxrregcount flag or the __launch_bounds__ qualifier to decrease the number of registers used by each thread. This will increase the number of blocks that can execute on each SM. On devices with Compute Capability 5.2 turning global cache off can increase the occupancy limited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

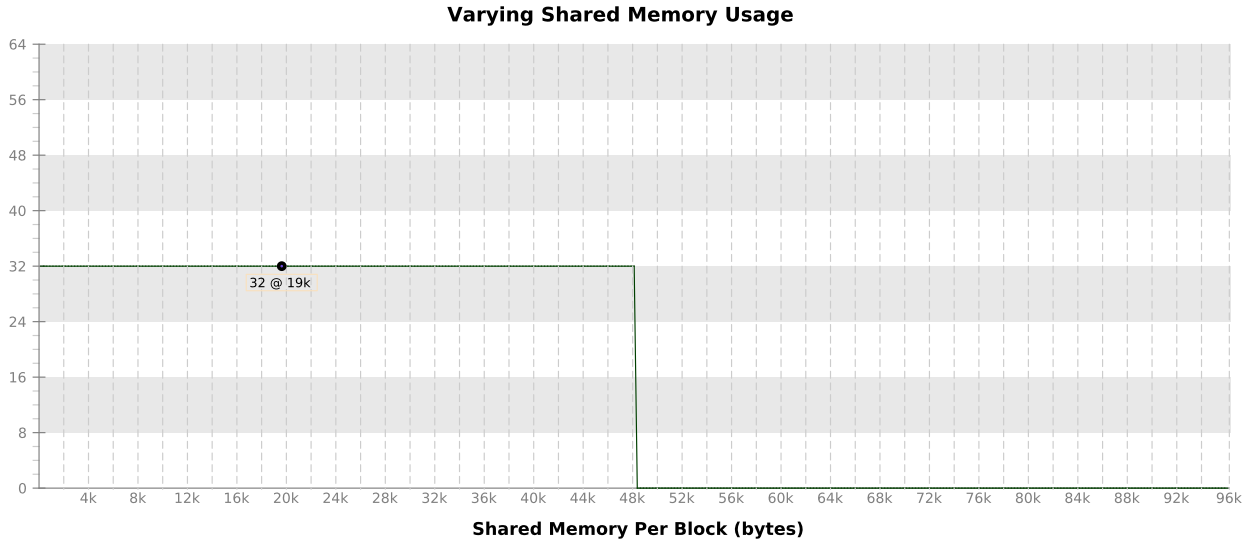
Variable	Achieved	Theoretical	Device Limit	Grid Size: [32,1,1] (32 blocks) Block Size: [32,32,1] (1024 threac
Occupancy Per SM				
Active Blocks		1	32	0 3 6 9 12 15 18 21 24 27 30 32
Active Warps	31.26	32	64	0 7 14 21 28 35 42 49 56 64
Active Threads		1024	2048	0 256 512 768 1024 1280 1536 1792 2048
Occupancy	48.8%	50%	100%	0% 25% 50% 75% 100%
Warps				
Threads/Block		1024	1024	0 128 256 384 512 640 768 896 1024
Warps/Block		32	32	0 3 6 9 12 15 18 21 24 27 30 32
Block Limit		2	32	0 3 6 9 12 15 18 21 24 27 30 32
Registers				
Registers/Thread		46	255	0 32 64 96 128 160 192 224 255
Registers/Block		49152	65536	0 16k 32k 48k 64k
Block Limit		1	32	0 3 6 9 12 15 18 21 24 27 30 32
Shared Memory				
Shared Memory/Block		20024	98304	0 32k 64k 96k
Block Limit		4	32	0 3 6 9 12 15 18 21 24 27 30 32

2.2. Occupancy Charts

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


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


Varying Register Count
64
56
48
40
32
24
16
8
0
32 @ 46
16
32
48
64
80
96
112
128
144
160
176
192
208
224
240
255
Registers Per Thread


Varying Shared Memory Usage
64
56
48
40
32
24
16
8
0
32 @ 19k
4k
8k
12k
16k
20k
24k
28k
32k
36k
40k
44k
48k
52k
56k
60k
64k
68k
72k
76k
80k
84k
88k
92k
96k
Shared Memory Per Block (bytes)

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 67.8% 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 68.3%.

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.

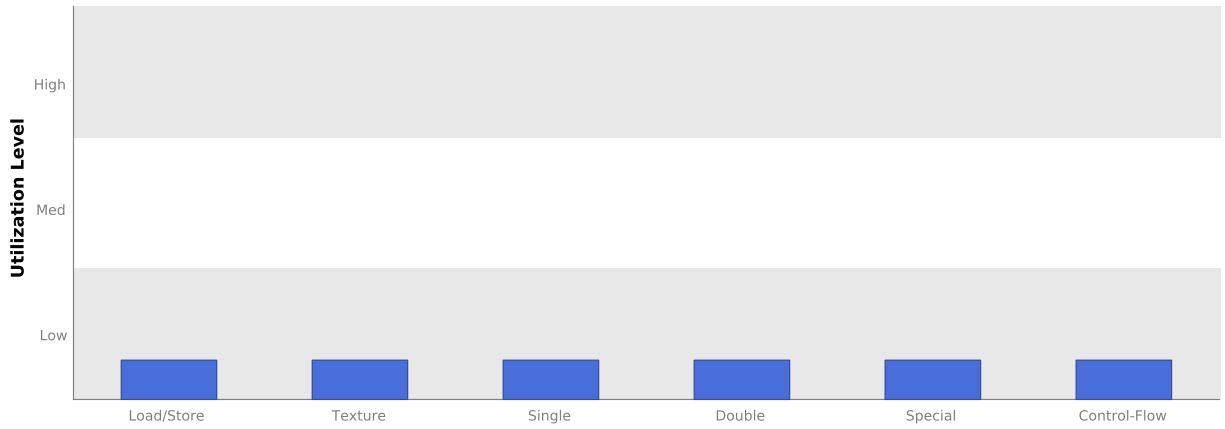
/home/i7426159/Desktop/Programming/Assignment/flockingSim/gpuApp/cudasrc/flockApp_gpu.cu

Line 257	Divergence = 90.6% [29 divergent executions out of 32 total executions]
Line 361	Divergence = 100% [32 divergent executions out of 32 total executions]
Line 464	Divergence = 81.2% [26 divergent executions out of 32 total executions]
Line 513	Divergence = 3.1% [32 divergent executions out of 1024 total executions]

3.3. Function Unit Utilization

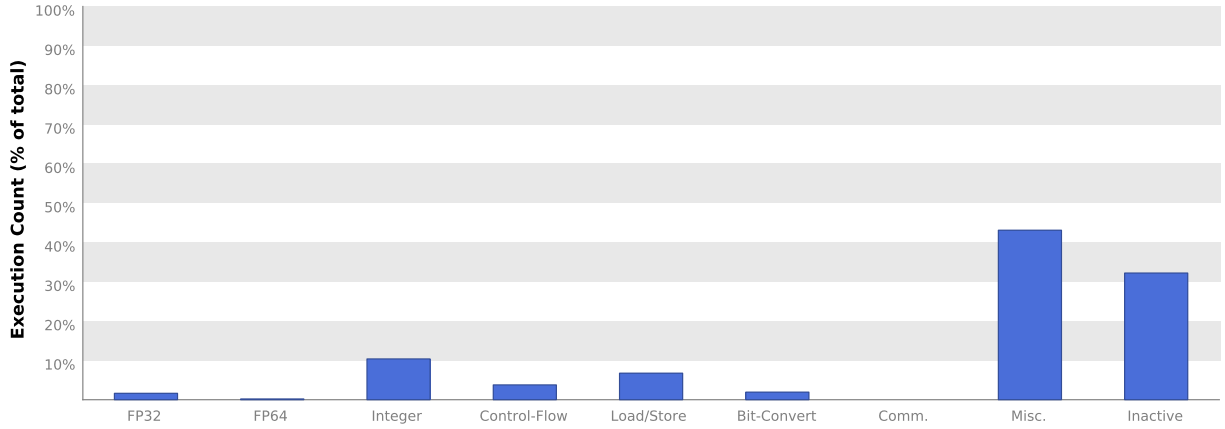
Different types of instructions are executed on different function units within each SM. Performance can be limited if a function unit is over-used by the instructions executed by the kernel. The following results show that the kernel's performance is not limited by overuse of any function unit.

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.

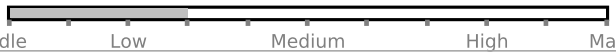
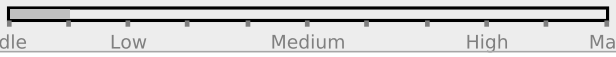

Execution Count (% of total)
100%
90%
80%
70%
60%
50%
40%
30%
20%
10%
FP32 Add
FP32 Mul
FP32 Mul-Add
FP Special
FP64 Add
FP64 Mul
FP64 Mul-Add

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.

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	10802	2.074 GB/s	
Shared Stores	12262	2.355 GB/s	
Shared Total	23064	4.429 GB/s	
L2 Cache			
Reads	319341	15.33 GB/s	
Writes	241454	11.591 GB/s	
Total	560795	26.921 GB/s	
Unified Cache			
Local Loads	437324	20.994 GB/s	
Local Stores	212546	10.203 GB/s	
Global Loads	100614	2.13 GB/s	
Global Stores	11000	528.054 MB/s	
Texture Reads	297823	14.297 GB/s	
Unified Total	1059307	48.152 GB/s	
Device Memory			
Reads	27901	1.339 GB/s	
Writes	106530	5.114 GB/s	
Total	134431	6.453 GB/s	
System Memory			
[PCIe configuration: Gen3 x16, 8 Gbit/s]			
Reads	0	0 B/s	
Writes	5	240.024 kB/s