

Analysis Report

avoidBoundaries_kernal(float*, float*, float*, float*, int)

Duration	31.04 μ s
Grid Size	[1,1,1]
Block Size	[1024,1,1]
Registers/Thread	27
Shared Memory/Block	0 B
Shared Memory Requested	96 KiB
Shared Memory Executed	96 KiB
Shared Memory Bank Size	4 B

[0] Quadro M4000

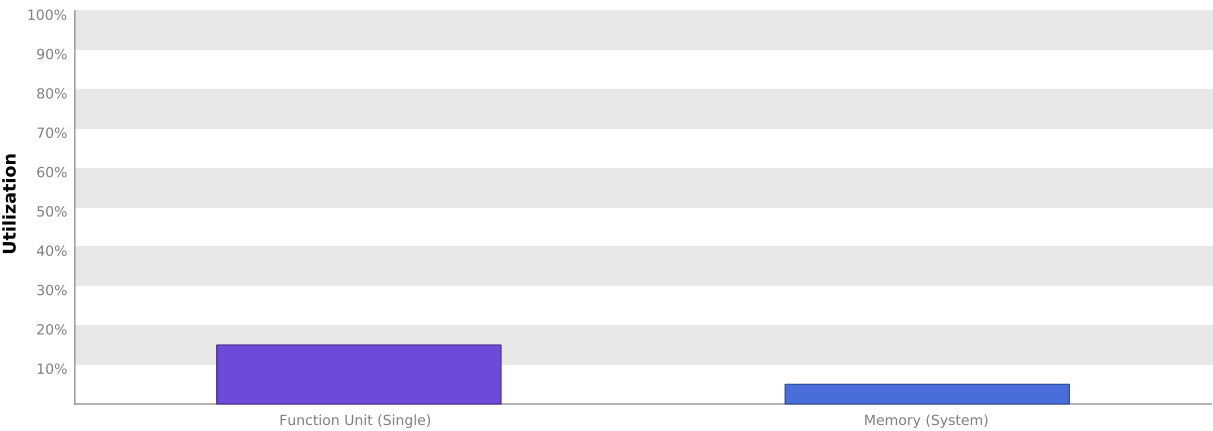
GPU UUID	GPU-45e1bfa2-9dfe-ee51-9716-d030e4dc476d
Compute Capability	5.2
Max. Threads per Block	1024
Max. Shared Memory per Block	48 KiB
Max. Registers per Block	65536
Max. Grid Dimensions	[2147483647, 65535, 65535]
Max. Block Dimensions	[1024, 1024, 64]
Max. Warps per Multiprocessor	64
Max. Blocks per 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 "avoidBoundaries_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 results below indicate that the GPU does not have enough work because the kernel does not execute enough blocks.

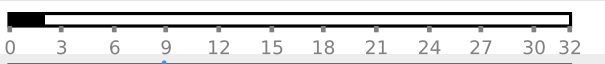
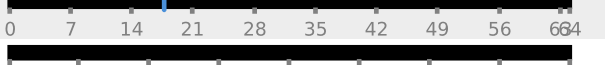
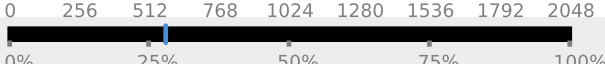


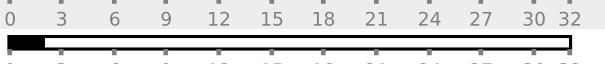
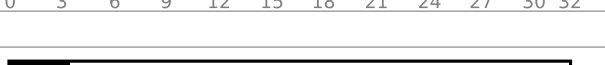
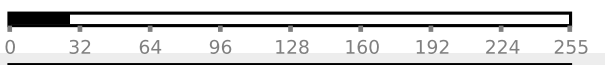

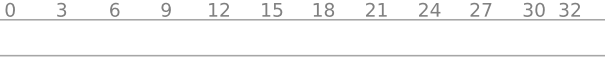
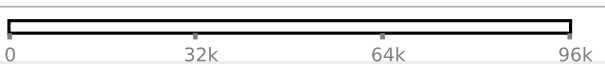
2.1. Grid Size Too Small To Hide Compute And Memory Latency

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 M4000" can simultaneously execute 2 blocks on each of the 13 SMs, so the kernel may need to execute a multiple of 26 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.

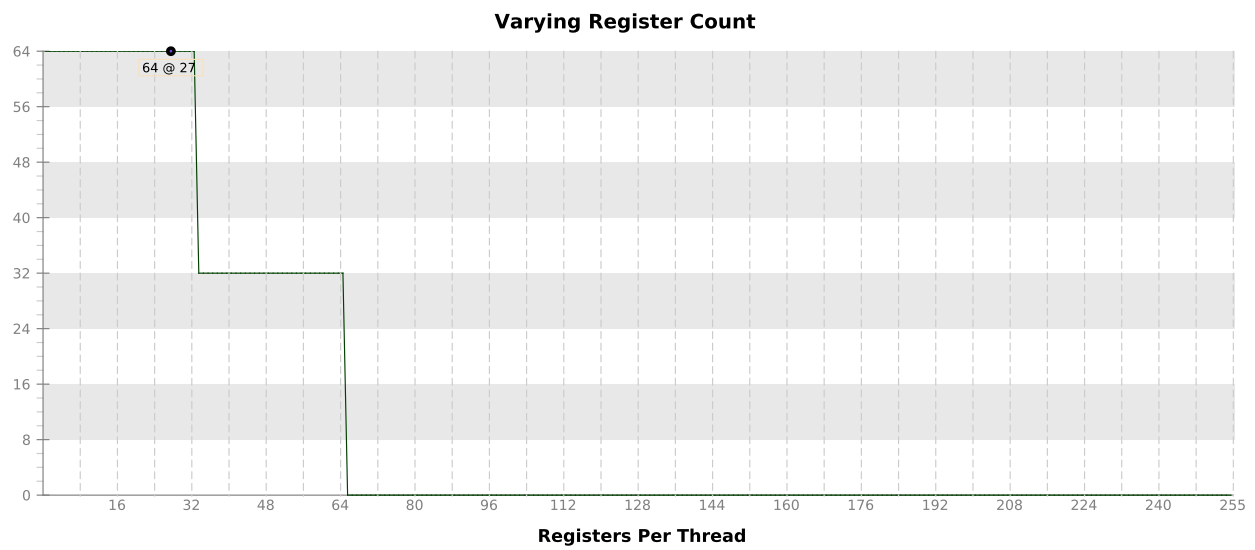
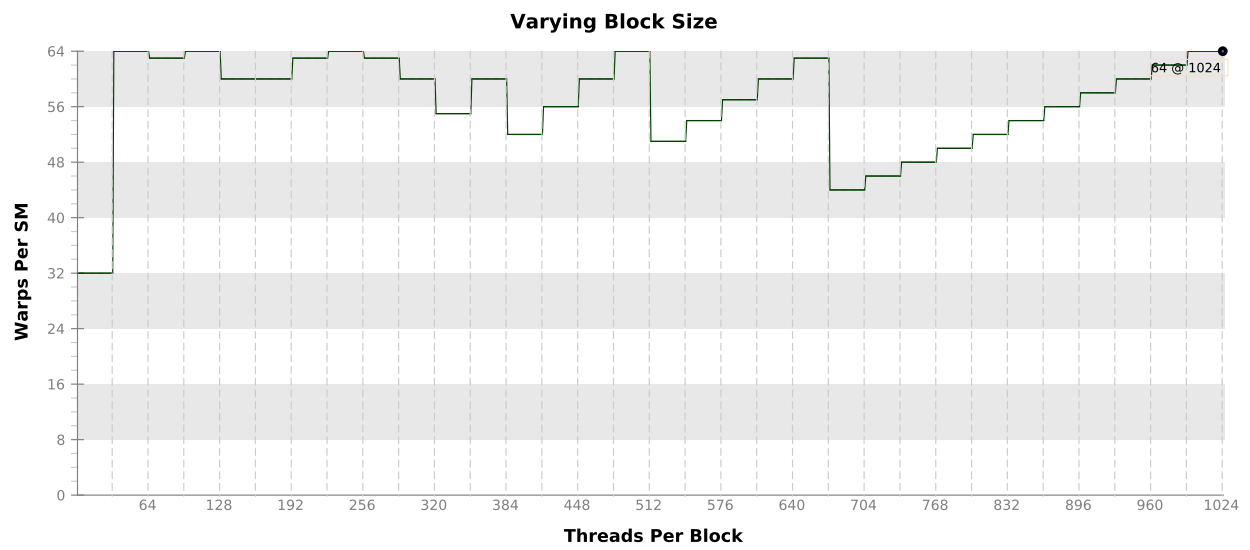
2.2. Occupancy Is Not Limiting Kernel Performance

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

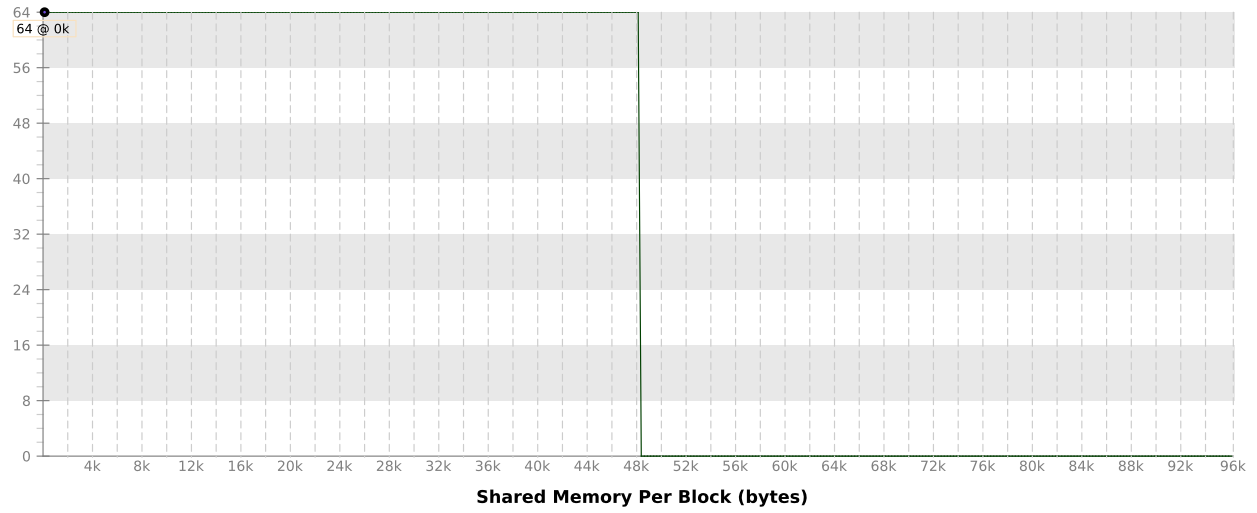
Variable	Achieved	Theoretical	Device Limit	Grid Size: [1,1,1] (1 block) Block Size: [1024,1,1] (1024 threads)
Occupancy Per SM				
Active Blocks		2	32	
Active Warps	17.47	64	64	
Active Threads		2048	2048	
Occupancy	27.3%	100%	100%	
Warps				
Threads/Block		1024	1024	
Warps/Block		32	32	
Block Limit		2	32	
Registers				
Registers/Thread		27	255	
Registers/Block		32768	65536	
Block Limit		2	32	
Shared Memory				
Shared Memory/Block		0	98304	
Block Limit			32	

2.3. 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. 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.

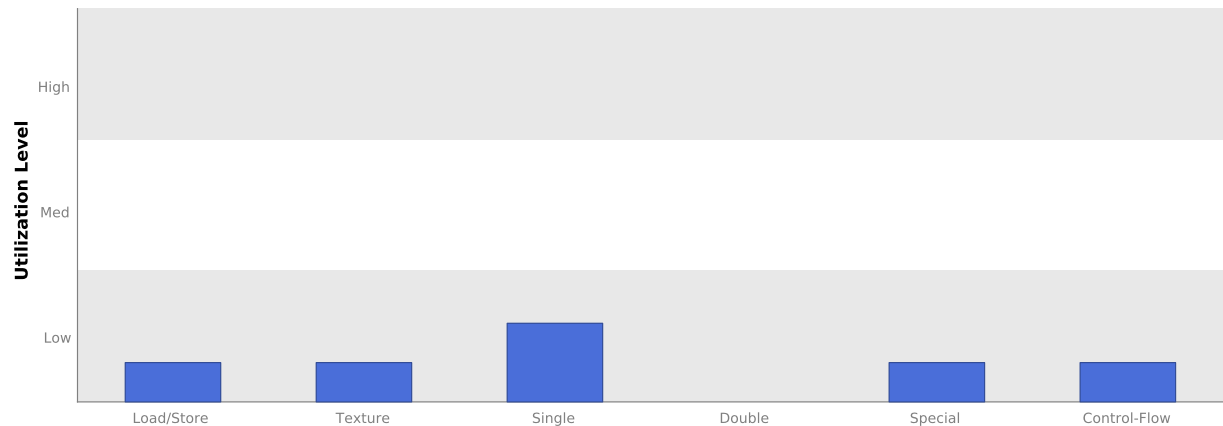
```
/home/i7426159/Desktop/Programming/Assignment/flockingSim/libflock_gpu/include/flock_kernels.cuh
```

Line 71	Divergence = 3.1% [1 divergent executions out of 32 total executions]
---------	---

3.2. Function Unit Utilization

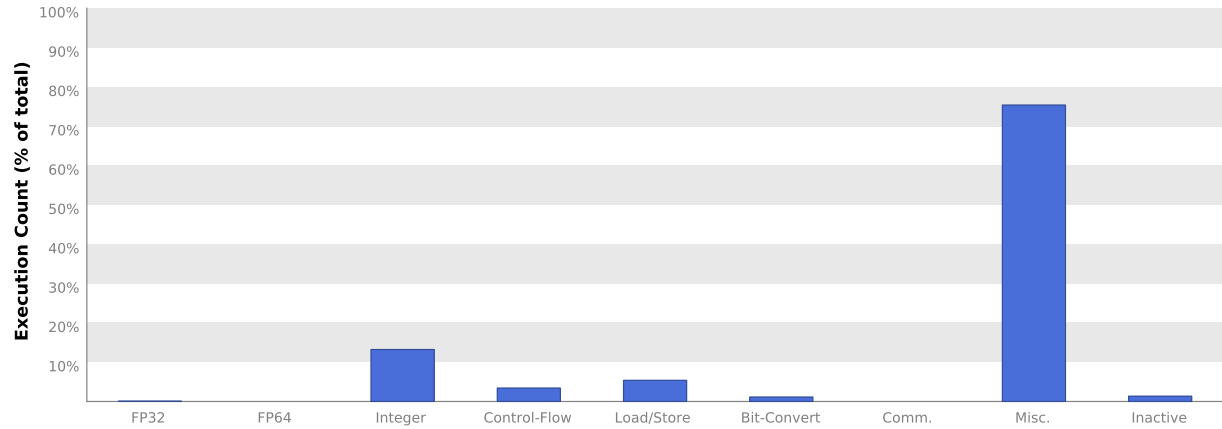
Different types of instructions are executed on different function units within each SM. Performance can be limited if a function unit is over-used by the instructions executed by the kernel. The following results show that the kernel's performance is not limited by overuse of any function unit.

- Load/Store - Load and store instructions for 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.



3.3. 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.



3.4. 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.



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	0	0 B/s	
Shared Stores	0	0 B/s	
Shared Total	0	0 B/s	
L2 Cache			
Reads	382	1.29 GB/s	
Writes	6	20.26 MB/s	
Total	388	1.31 GB/s	
Unified Cache			
Local Loads	0	0 B/s	
Local Stores	0	0 B/s	
Global Loads	20	40.519 MB/s	
Global Stores	0	0 B/s	
Texture Reads	12	40.519 MB/s	
Unified Total	32	81.038 MB/s	
Device Memory			
Reads	132	445.711 MB/s	
Writes	4	13.506 MB/s	
Total	136	459.217 MB/s	
System Memory			
[PCIe configuration: Gen3 x16, 8 Gbit/s]			
Reads	0	0 B/s	
Writes	5	16.883 MB/s	