Analysis Report

void computeROIwarpReadOnly<float, int=64, int=7, int=15>(float const *, float*, float const *, float, unsigned int, unsigned int)

Duration	1.811 ms (1,811,039 ns)
Grid Size	[1947,1,1]
Block Size	[256,1,1]
Registers/Thread	32
Shared Memory/Block	0 B
Shared Memory Requested	96 KiB
Shared Memory Executed	96 KiB
Shared Memory Bank Size	4 B

[0] GeForce GTX 960

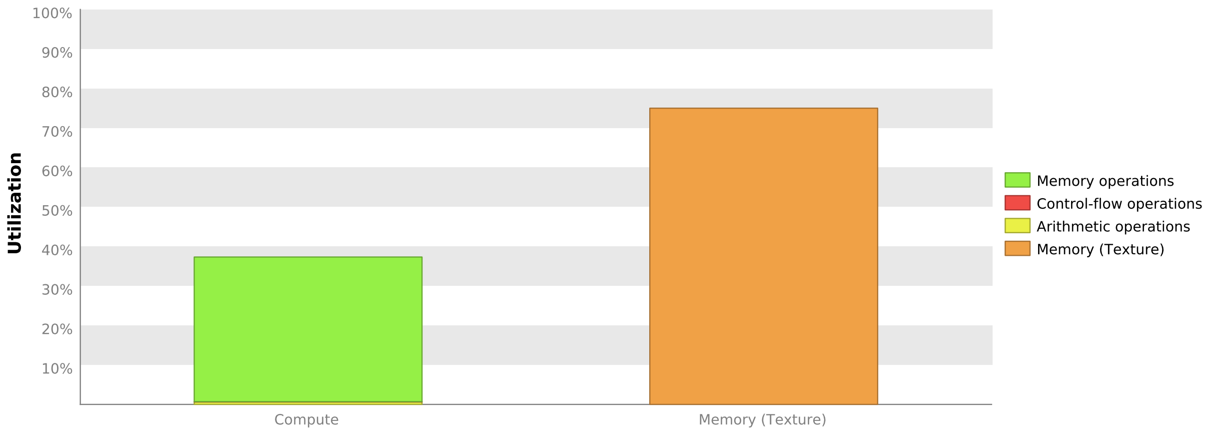
GPU UUID	GPU-0db32734-f94e-48a7-8b5d-4604317dc554
Compute Capability	5.2
Max. Threads per Block	1024
Max. Shared Memory per Block	48 KiB
Max. Registers per Block	65536
Max. Grid Dimensions	[2147483647, 65535, 65535]
Max. Block Dimensions	[1024, 1024, 64]
Max. Warps per Multiprocessor	64
Max. Blocks per Multiprocessor	32
Single Precision FLOP/s	2.644 TeraFLOP/s
Double Precision FLOP/s	82.624 GigaFLOP/s
Number of Multiprocessors	8
Multiprocessor Clock Rate	1.291 GHz
Concurrent Kernel	true
Max IPC	6
Threads per Warp	32
Global Memory Bandwidth	112.16 GB/s
Global Memory Size	4 GiB
Constant Memory Size	64 KiB
L2 Cache Size	1 MiB
Memcpy Engines	2
PCIe Generation	2
PCIe Link Rate	5 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 "void computeROIwarpReadOnly..." 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 GTX 960" 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 Texture 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 L2 cache.

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 memories with high bandwidth utilization.
Shared Memory - If possible use 64-bit accesses to shared memory and 8-byte bank mode to achieved 2x throughput.
L2 Cache - Align and block kernel data to maximize L2 cache efficiency.
Unified Cache - Reallocate texture data to shared or global memory. Resolve alignment and access pattern issues for global loads and stores.
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	
Shared Memory			
Shared Loads	0	0 B/s	
Shared Stores	0	0 B/s	
Shared Total	0	0 B/s	Idle Low Medium High Max
L2 Cache			
Reads	14403679	274.422 GB/s	
Writes	15580	296.834 MB/s	
Total	14419259	274.719 GB/s	Idle Low Medium High Max
Unified Cache			
Local Loads	0	0 B/s	
Local Stores	0	0 B/s	
Global Loads	52328640	747.734 GB/s	
Global Stores	15574	296.72 MB/s	
Texture Reads	26164320	498.489 GB/s	
Unified Total	78508534	1,246.519 GB/s	Idle Low Medium High Max
Device Memory			
Reads	144567	2.754 GB/s	
Writes	17605	335.415 MB/s	
Total	162172	3.09 GB/s	Idle Low Medium High Max
System Memory			
[PCIe configuration: Gen2 x16, 5 Gbit/s]			
Reads	0	0 B/s	Idle Low Medium High Max
Writes	5	95.261 kB/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.

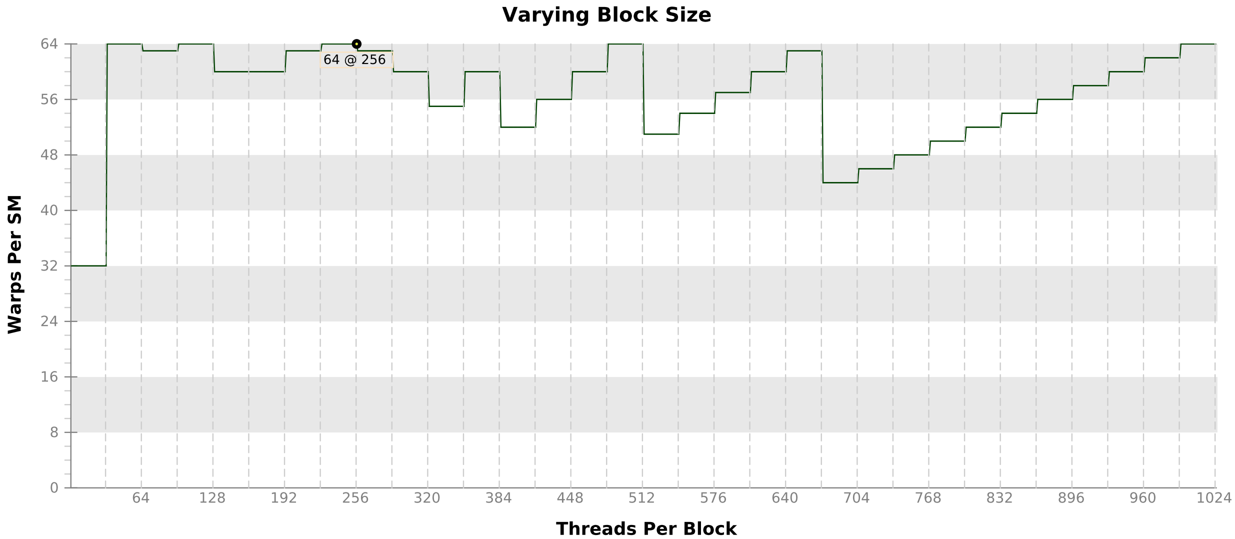
3.1. Occupancy Is Not Limiting Kernel Performance

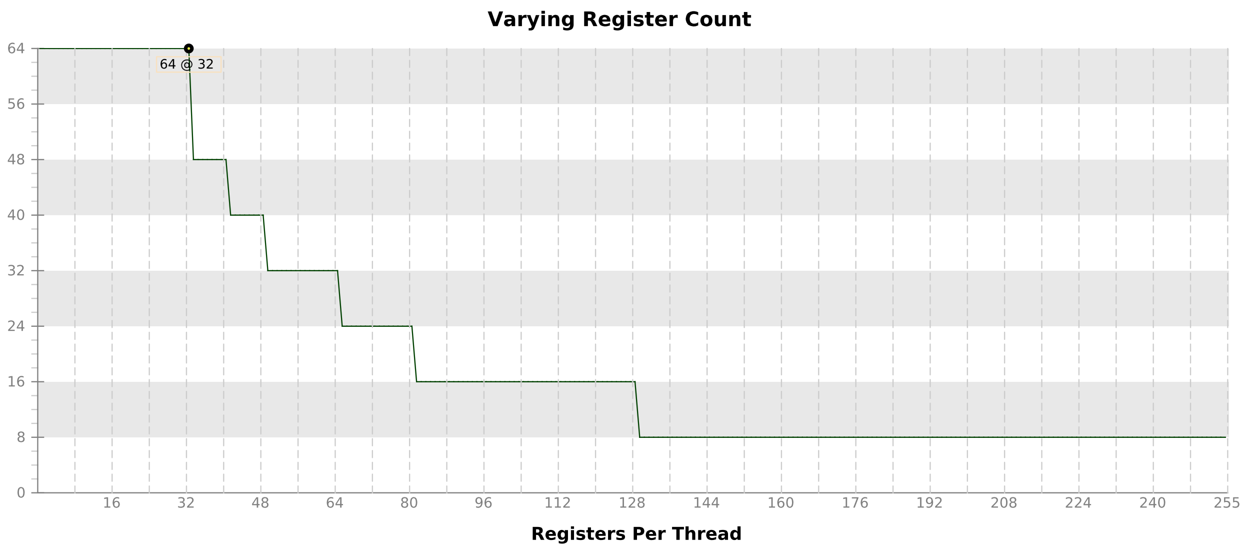
The kernel's block size, register usage, and shared memory usage allow it to fully utilize all warps on the GPU.

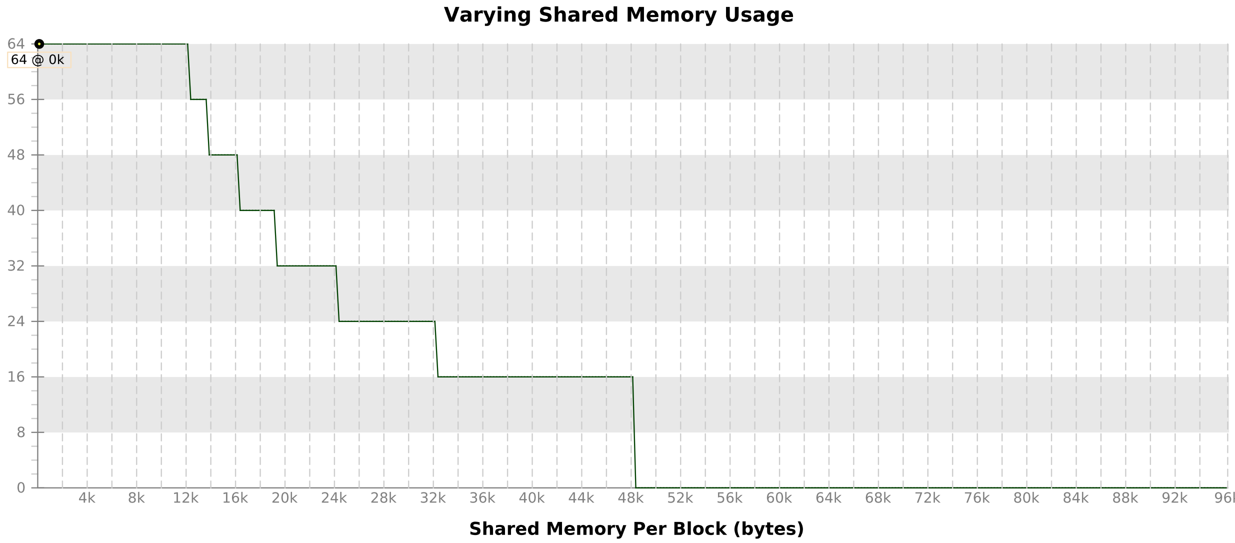
Variable	Achieved	Theoretical	Device Limit	Grid Size: [1947,1,1] (1947 blocks) Block Size: [256,1,1] (256 th
Occupancy Per SM				
Active Blocks		8	32	
Active Warps	62.84	64	64	
Active Threads		2048	2048	
Occupancy	98.2%	100%	100%	
Warps				
Threads/Block		256	1024	
Warps/Block		8	32	
Block Limit		8	32	
Registers				
Registers/Thread		32	255	
Registers/Block		8192	65536	
Block Limit		8	32	
Shared Memory				
Shared Memory/Block		0	98304	
Block Limit			32	

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
Threads Per Block
64 @ 256
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
64 @ 32
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
64
56
48
40
32
24
16
8
0
64 @ 0k
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)

4. 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.

4.1. 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.

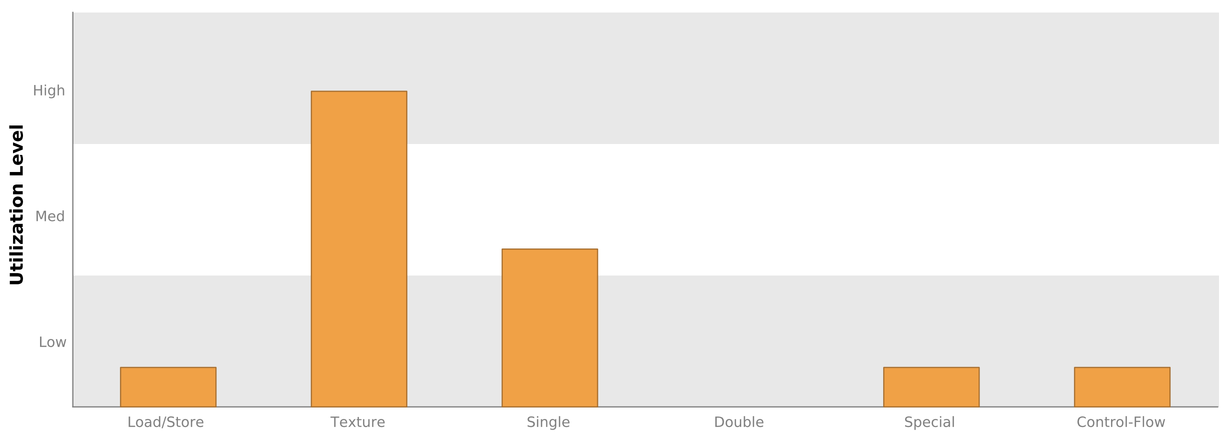
Optimization: Each entry below points to a divergent branch within the kernel. For each branch reduce the amount of intra-warp divergence.

	/home/adas/cuda-workspace/CudaVisionSysDeploy/Release/../src/init/../device/SVM/SVMclassification.h
Line 42	Divergence = 100% [15574 divergent executions out of 15574 total executions]

4.2. Function Unit Utilization

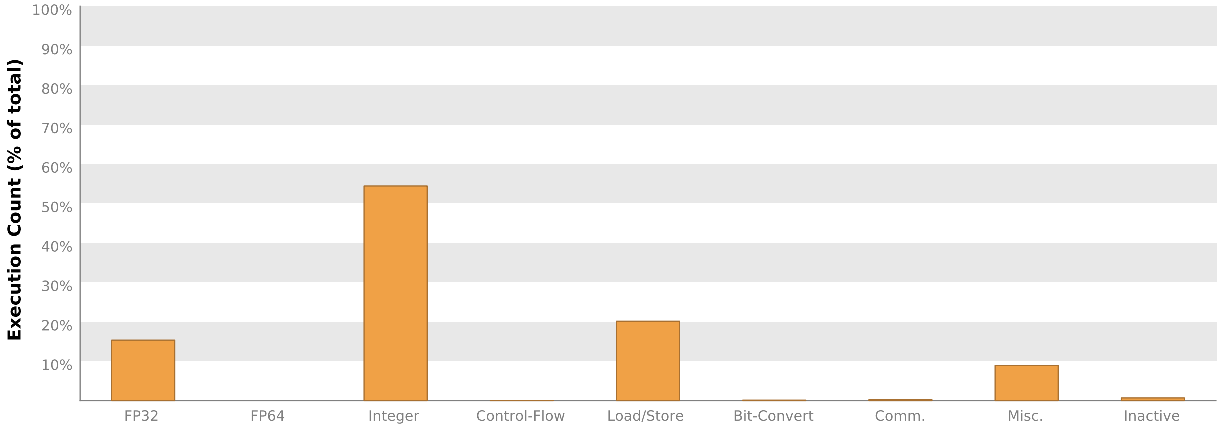
Different types of instructions are executed on different function units within each SM. Performance can be limited if a function unit is over-used by the instructions executed by the kernel. The following results show that the kernel's performance is not limited by overuse of any function unit.

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.



4.3. 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.



4.4. 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.