

Analysis Report

assignClassUsingMeanVarianceDataUsingFeatureSelectionKernel(float*, float*, int*, int, int, int, int, int*, int*)

Duration	8.539 ms (8,539,204 ns)
Grid Size	[209,1,1]
Block Size	[64,1,1]
Registers/Thread	36
Shared Memory/Block	0 B
Shared Memory Requested	48 KiB
Shared Memory Executed	48 KiB
Shared Memory Bank Size	4 B

[0] Quadro K2000

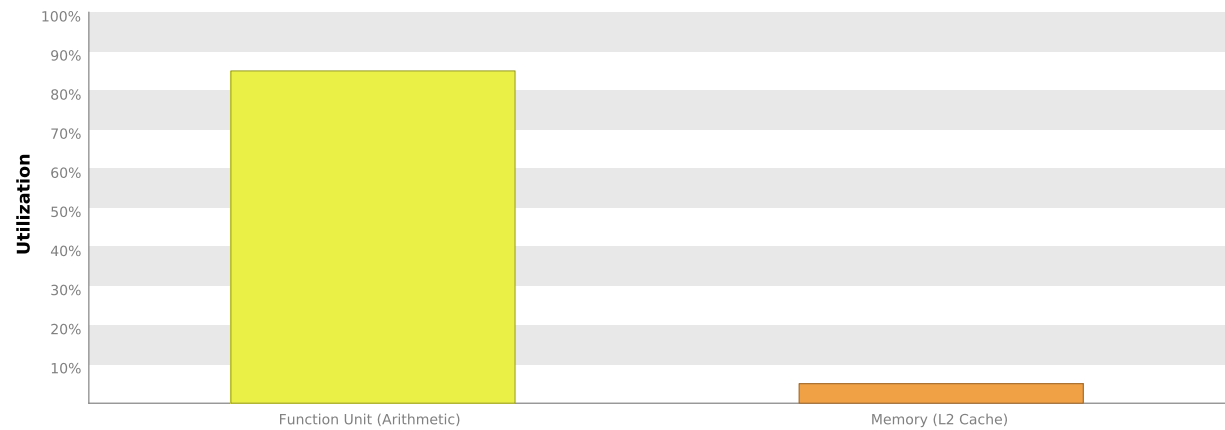
GPU UUID	GPU-c7ae453e-7cbe-1397-56a9-319a9ce07467
Compute Capability	3.0
Max. Threads per Block	1024
Max. Shared Memory per Block	48 KiB
Max. Registers per Block	65536
Max. Grid Dimensions	[2147483647, 65535, 65535]
Max. Block Dimensions	[1024, 1024, 64]
Max. Warps per Multiprocessor	64
Max. Blocks per Multiprocessor	16
Single Precision FLOP/s	732.672 GigaFLOP/s
Double Precision FLOP/s	30.528 GigaFLOP/s
Number of Multiprocessors	2
Multiprocessor Clock Rate	954 MHz
Concurrent Kernel	true
Max IPC	7
Threads per Warp	32
Global Memory Bandwidth	64 GB/s
Global Memory Size	1.996 GiB
Constant Memory Size	64 KiB
L2 Cache Size	256 KiB
Memcpy Engines	1
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 "assignClassUsingMeanVarianc..." 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 "Quadro K2000" 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. Compute resources are used most efficiently when instructions do not overuse a function unit. The results below indicate that compute performance may be limited by overuse of a function unit.

2.1. Kernel Profile - Instruction Execution

The Kernel Profile - Instruction Execution shows the execution count, inactive threads, and predicated threads for each source and assembly line of the kernel. Using this information you can pinpoint portions of your kernel that are making inefficient use of compute resource due to divergence and predication.

Examine portions of the kernel that have high execution counts and inactive or predicated threads to identify optimization opportunities.

Cuda Functions :

<code>assignClassUsingMeanVarianceDataUsingFeatureSelectionKernel(float*, float*, int*, int, int, int, int, int*, int*)</code>
--

Maximum instruction execution count in assembly: 156737

Average instruction execution count in assembly: 28250

Instructions executed for the kernel: 17402221

Thread instructions executed for the kernel: 243650842

Non-predicated thread instructions executed for the kernel: 234820037

Warp non-predicated execution efficiency of the kernel: 42.2%

Warp execution efficiency of the kernel: 43.8%

2.2. 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.

2.3. GPU Utilization Is Limited By Function Unit Usage

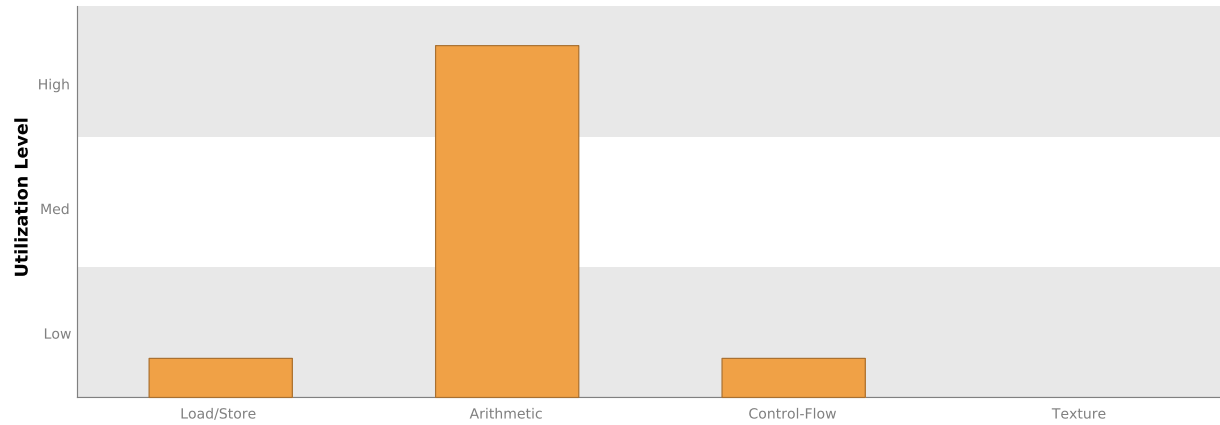
Different types of instructions are executed on different function units within each SM. Performance can be limited if a function unit is over-used by the instructions executed by the kernel. The following results show that the kernel's performance is potentially limited by overuse of the following function units: Arithmetic.

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.

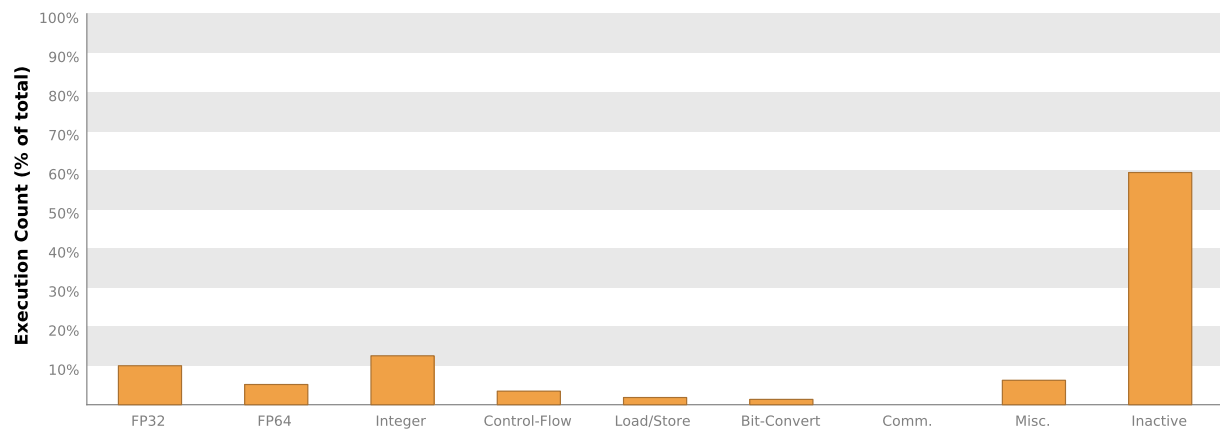
Control-Flow - Direct and indirect branches, jumps, and calls.

Texture - Texture operations.



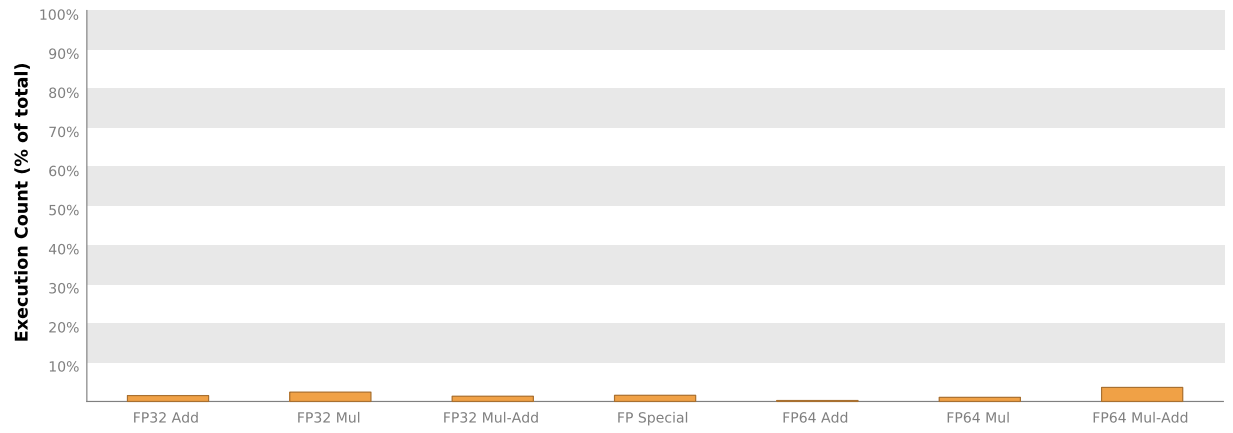
2.4. Instruction Execution Counts

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.



2.5. 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.



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. Unfortunately, the device executing this kernel can not provide the profile data needed for this analysis.

4. Instruction and Memory Latency





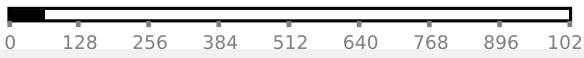
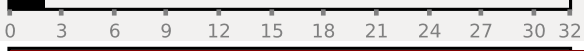





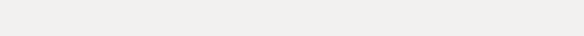
Instruction and memory latency limit the performance of a kernel when the GPU does not have enough work to keep busy. The performance of latency-limited kernels can often be improved by increasing occupancy. Occupancy is a measure of how many warps the kernel has active on the GPU, relative to the maximum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

4.1. GPU Utilization May Be Limited By Block Size

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 has a block size of 64 threads. This block size is likely preventing the kernel from fully utilizing the GPU. Device "Quadro K2000" can simultaneously execute up to 16 blocks on each SM. Because each block uses 2 warps to execute the block's 64 threads, the kernel is using only 32 warps on each SM. Chart "Varying Block Size" below shows how changing the block size will change the number of warps that can execute on each SM.

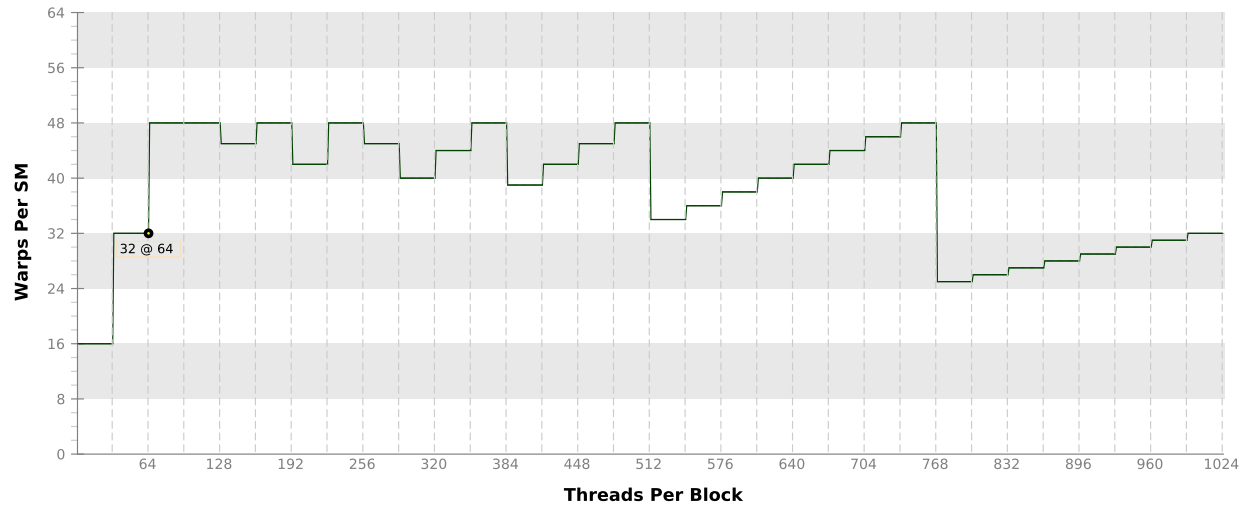
Optimization: Increase the number of threads in each block to increase the number of warps that can execute on each SM.

Variable	Achieved	Theoretical	Device Limit	Grid Size: [209,1,1] (209 blocks) Block Size: [64,1,1] (64 threads)
Occupancy Per SM				
Active Blocks		16	16	
Active Warps	29.93	32	64	
Active Threads		1024	2048	
Occupancy	46.8%	50%	100%	
Warps				
Threads/Block		64	1024	
Warps/Block		2	32	
Block Limit		32	16	
Registers				
Registers/Thread		36	63	
Registers/Block		2560	65536	
Block Limit		24	16	
Shared Memory				
Shared Memory/Block		0	49152	
Block Limit			16	

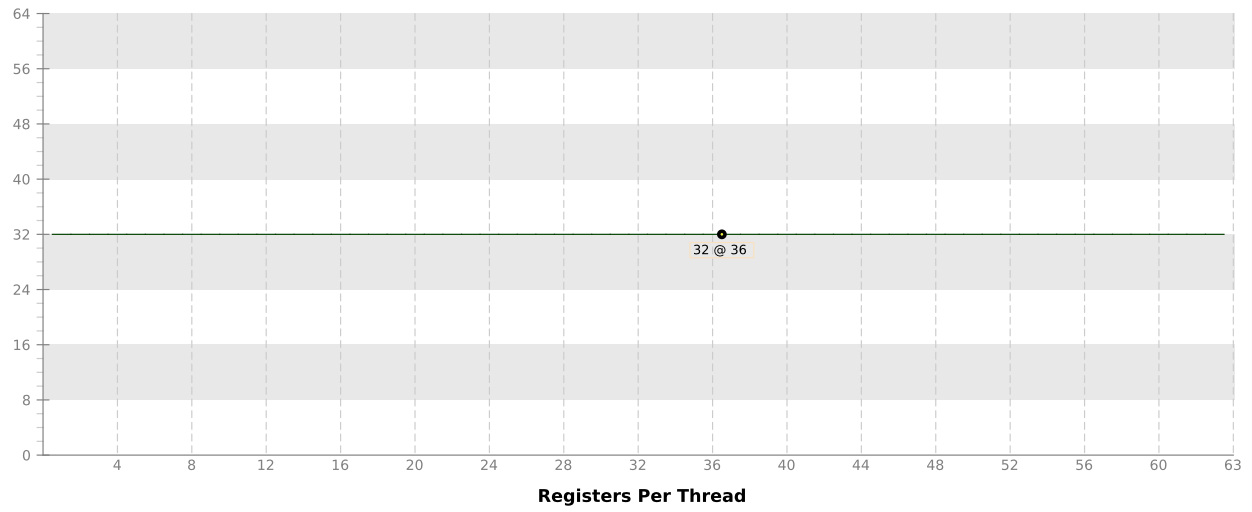
4.2. Occupancy Charts

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

Varying Block Size



Varying Register Count



Varying Shared Memory Usage

