

Analysis Report

KNNQuery_base(float*, float*, float*, float*, float*, float*, Point2Rep*, Point2Rep*, repPoint_static_dev*, repPoint_dynamic_p*, repPoint_static_dev*, repPoint_dynamic_p*, int, int, int, int, int, int, IndexAndDist*, int*)

Duration	4.274 s (4,273,715,265 ns)
Grid Size	[567,1,1]
Block Size	[256,1,1]
Registers/Thread	51
Shared Memory/Block	0 B
Shared Memory Requested	48 KiB
Shared Memory Executed	48 KiB
Shared Memory Bank Size	4 B

[0] Tesla K20c

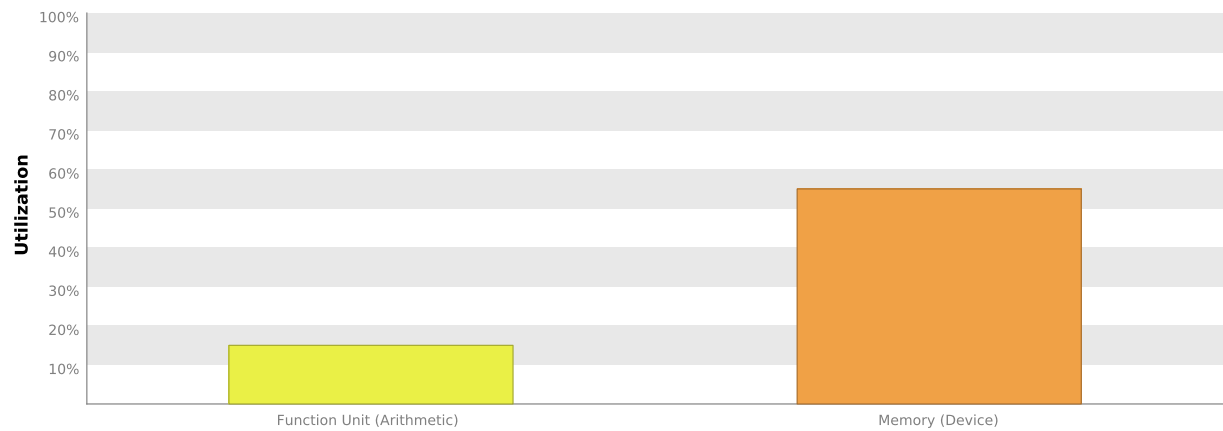
GPU UUID	GPU-5af13825-b086-5bfe-f175-a239faf0f1f9
Compute Capability	3.5
Max. Threads per Block	1024
Max. Shared Memory per Block	48 KiB
Max. Registers per Block	65536
Max. Grid Dimensions	[2147483647, 65535, 65535]
Max. Block Dimensions	[1024, 1024, 64]
Max. Warps per Multiprocessor	64
Max. Blocks per Multiprocessor	16
Single Precision FLOP/s	3.522 TeraFLOP/s
Double Precision FLOP/s	1.174 TeraFLOP/s
Number of Multiprocessors	13
Multiprocessor Clock Rate	705.5 MHz
Concurrent Kernel	true
Max IPC	7
Threads per Warp	32
Global Memory Bandwidth	208 GB/s
Global Memory Size	4.687 GiB
Constant Memory Size	64 KiB
L2 Cache Size	1.25 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 "KNNQuery_base" 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 "Tesla K20c". 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. Unfortunately, the device executing this kernel can not provide the profile data needed for this analysis.

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 36.8% 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 44.3%.

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.

3.3. Function Unit Utilization

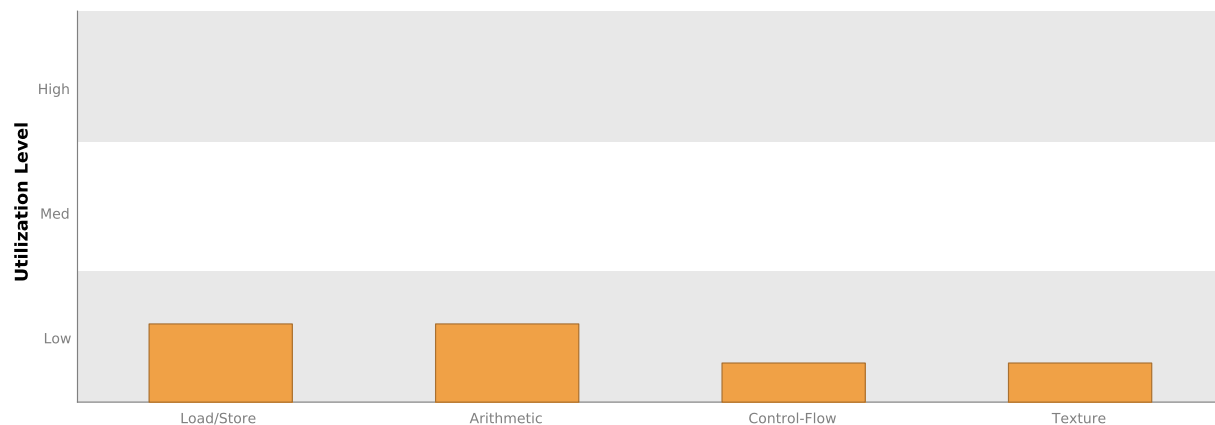
Different types of instructions are executed on different function units within each SM. Performance can be limited if a function unit is over-used by the instructions executed by the kernel. The following results show that the kernel's performance is not limited by overuse of any function unit.

Load/Store - Load and store instructions for 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.

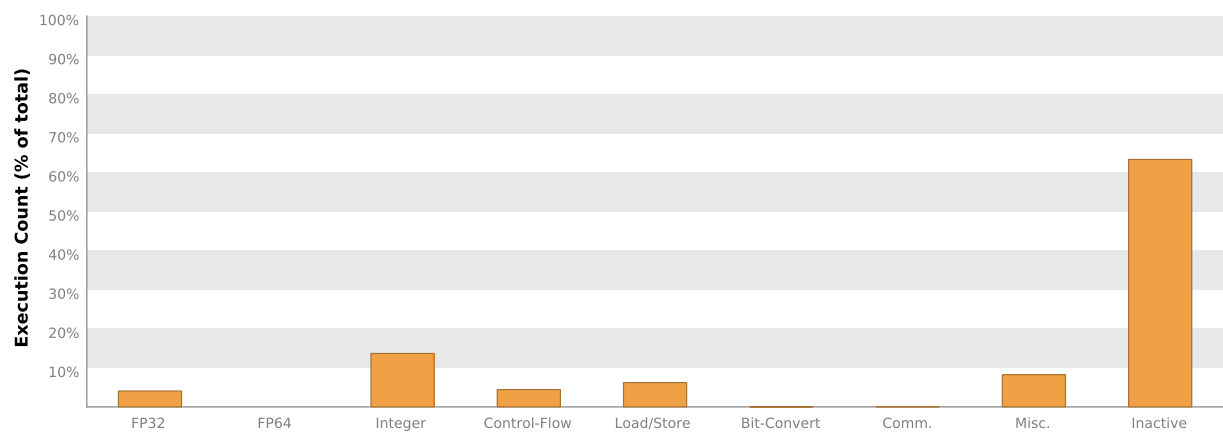
Texture - Texture operations.



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.



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. The results below indicate that the kernel is limited by the bandwidth available to the device memory.

4.1. Global Memory Alignment and Access Pattern

Memory bandwidth is used most efficiently when each global memory load and store has proper alignment and access pattern.

Optimization: Each entry below points to a global load or store within the kernel with an inefficient alignment or access pattern. For each load or store improve the alignment and access pattern of the memory access.





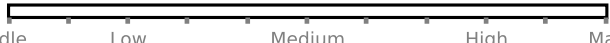
4.2. High Local Memory Overhead

Local memory loads and stores account for 97% of total memory traffic. High local memory traffic typically indicates excessive register spilling.

Optimization: Use the -maxrregcount flag or the __launch_bounds__ qualifier to increase the number of registers available to nvcc when compiling the kernel.

4.3. 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	2267018600	67.721 GB/s	
Local Stores	719067384	19.259 GB/s	
Shared Loads	0	0 B/s	
Shared Stores	0	0 B/s	
Global Loads	118442657	1.123 GB/s	
Global Stores	232091200	1.749 GB/s	
Atomic	73170667	73.171 MB/s	
L1/Shared Total	3409790508	89.925 GB/s	
L2 Cache			
L1 Reads	9099098301	68.579 GB/s	
L1 Writes	2883940041	21.736 GB/s	
Texture Reads	6462704	48.709 MB/s	
Noncoherent Reads	6454064	48.643 MB/s	
Atomic	141469230	533.118 MB/s	
Total	11989501046	90.363 GB/s	
Texture Cache			
Reads	96167246	724.801 MB/s	
Device Memory			
Reads	11252770327	84.811 GB/s	
Writes	3375993835	25.444 GB/s	
Total	14628764162	110.255 GB/s	
ECC Overhead	3039197387	22.906 GB/s	
System Memory			
[PCIe configuration: Gen2 x16, 5 Gbit/s]			
Reads	0	0 B/s	
Writes	0	0 B/s	