Analysis Report

jac_gpu2(int, double, int, double*, double*, double*)

Duration	129.151 µs
Grid Size	[129,129,1]
Block Size	[16,16,1]
Registers/Thread	26
Shared Memory/Block	0 B
Shared Memory Requested	96 KiB
Shared Memory Executed	96 KiB
Shared Memory Bank Size	4 B

[0] Tesla V100-PCIE-16GB

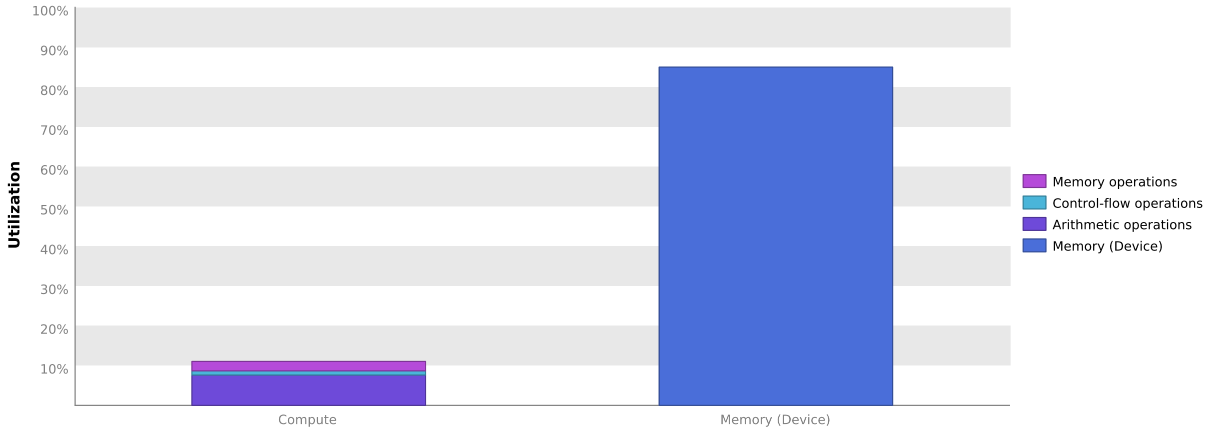
GPU UUID	GPU-fe492529-48c2-9a7f-92d9-a7ace94a08d9
Compute Capability	7.0
Max. Threads per Block	1024
Max. Threads per Multiprocessor	2048
Max. Shared Memory per Block	48 KiB
Max. Shared Memory per Multiprocessor	96 KiB
Max. Registers per Block	65536
Max. Registers per Multiprocessor	65536
Max. Grid Dimensions	[2147483647, 65535, 65535]
Max. Block Dimensions	[1024, 1024, 64]
Max. Warps per Multiprocessor	64
Max. Blocks per Multiprocessor	32
Half Precision FLOP/s	28.262 TeraFLOP/s
Single Precision FLOP/s	14.131 TeraFLOP/s
Double Precision FLOP/s	7.066 TeraFLOP/s
Number of Multiprocessors	80
Multiprocessor Clock Rate	1.38 GHz
Concurrent Kernel	true
Max IPC	4
Threads per Warp	32
Global Memory Bandwidth	898.048 GB/s
Global Memory Size	15.774 GiB
Constant Memory Size	64 KiB
L2 Cache Size	6 MiB
Memcpy Engines	7
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 "jac_gpu2" 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 "Tesla V100-PCIE-16GB" 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 Device 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 device memory.

2.1. 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. The analysis is per assembly instruction.

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.

2.2. 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 memory 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	2772249	685.183 GB/s	
Writes	1265852	312.865 GB/s	
Total	4038101	998.048 GB/s	Idle Low Medium High Max
Unified Cache			
Local Loads	0	0 B/s	
Local Stores	0	0 B/s	
Global Loads	6168285	1,524.551 GB/s	
Global Stores	1181696	292.065 GB/s	
Texture Reads	1709839	1,690.399 GB/s	
Unified Total	9059820	3,507.016 GB/s	Idle Low Medium High Max
Device Memory			
Reads	2109241	521.315 GB/s	
Writes	1133962	280.267 GB/s	
Total	3243203	801.583 GB/s	Idle Low Medium High Max
System Memory			
[PCIe configuration: Gen3 x16, 8 Gbit/s]			
Reads	0	0 B/s	Idle Low Medium High Max
Writes	5	1.236 MB/s	Idle Low Medium High Max

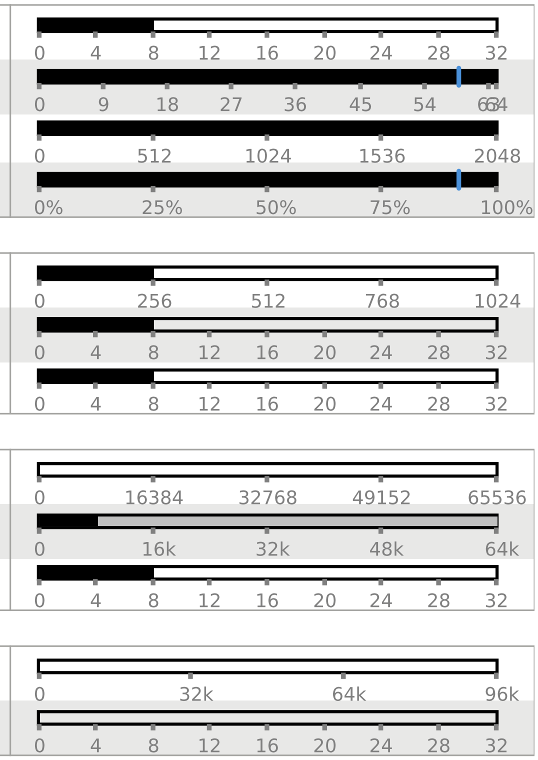
3. Instruction and Memory Latency

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

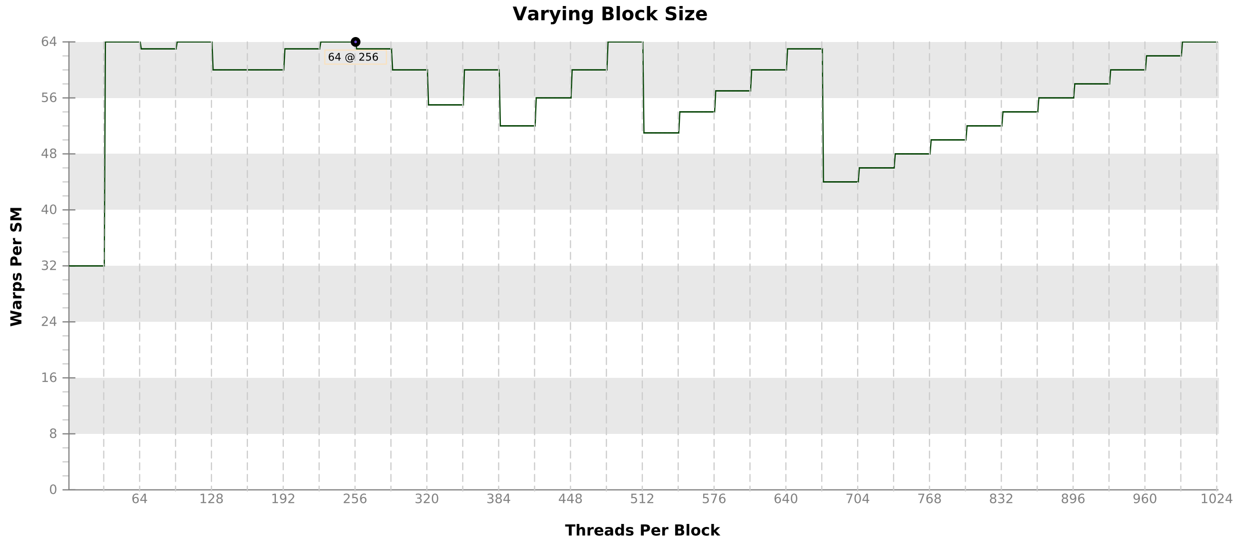
The kernel's block size, register usage, and shared memory usage allow it to fully utilize all warps on the GPU.

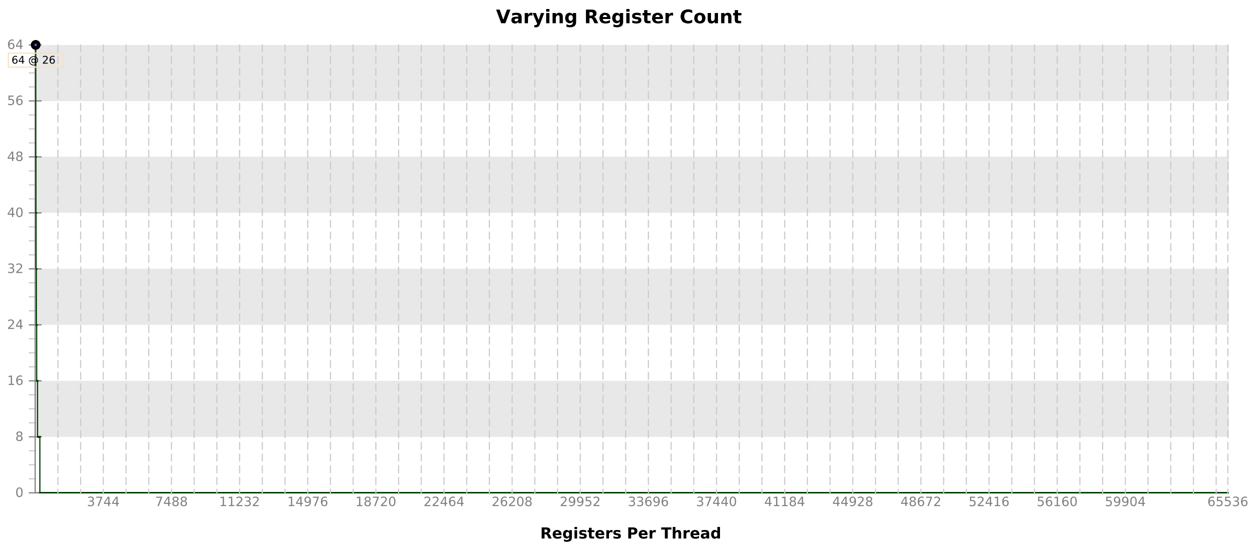
Variable	Achieved	Theoretical	Device Limit	Grid Size: [129,129,1] (16641 blocks) Block Size: [16,1
Occupancy Per SM				
Active Blocks		8	32	
Active Warps	58.34	64	64	
Active Threads		2048	2048	
Occupancy	91.2%	100%	100%	
Warps				
Threads/Block		256	1024	
Warps/Block		8	32	
Block Limit		8	32	
Registers				
Registers/Thread		26	65536	
Registers/Block		8192	65536	
Block Limit		8	32	
Shared Memory				
Shared Memory/Block		0	98304	
Block Limit		0	32	

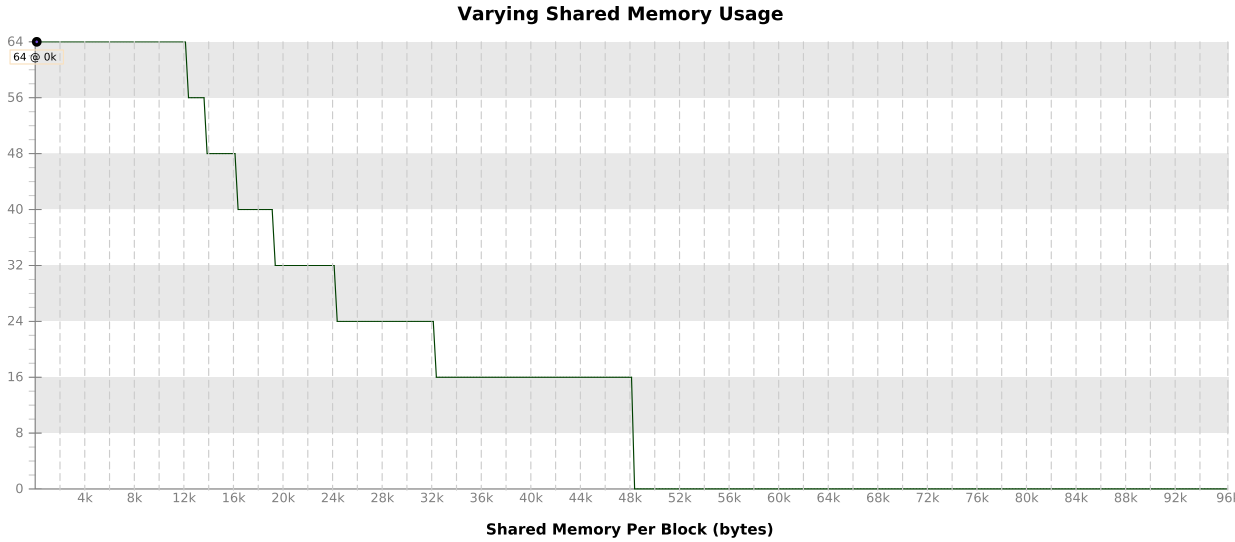


3.2. Occupancy Charts

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

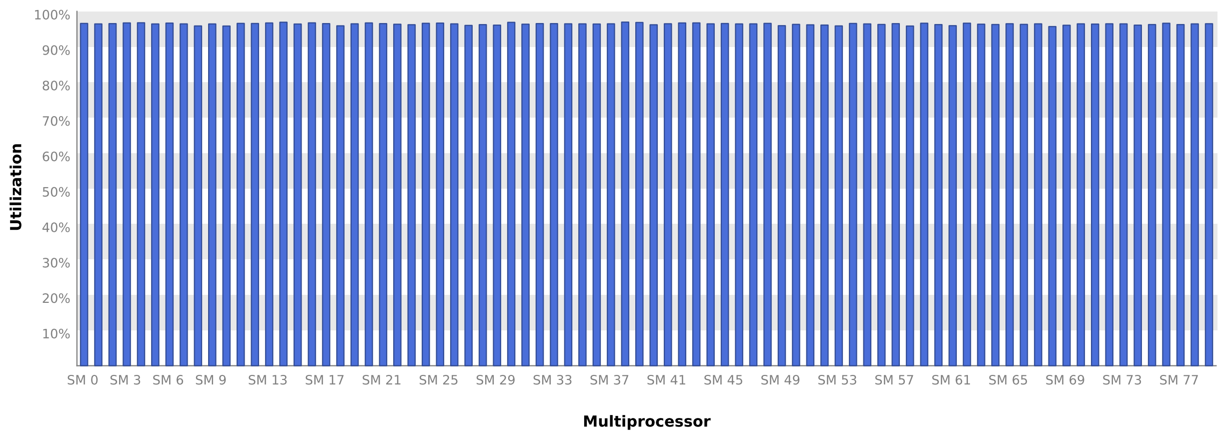

Varying Block Size
Warps Per SM
Threads Per Block
64 @ 256
0
8
16
24
32
40
48
56
64
128
192
256
320
384
448
512
576
640
704
768
832
896
960
1024


Varying Register Count
Registers Per Thread
64 @ 26
0
8
16
24
32
40
48
56
64
3744
7488
11232
14976
18720
22464
26208
29952
33696
37440
41184
44928
48672
52416
56160
59904
65536



3.3. Multiprocessor Utilization

The kernel's blocks are distributed across the GPU's multiprocessors for execution. Depending on the number of blocks and the execution duration of each block some multiprocessors may be more highly utilized than others during execution of the kernel. The following chart shows the utilization of each multiprocessor during execution of the kernel.



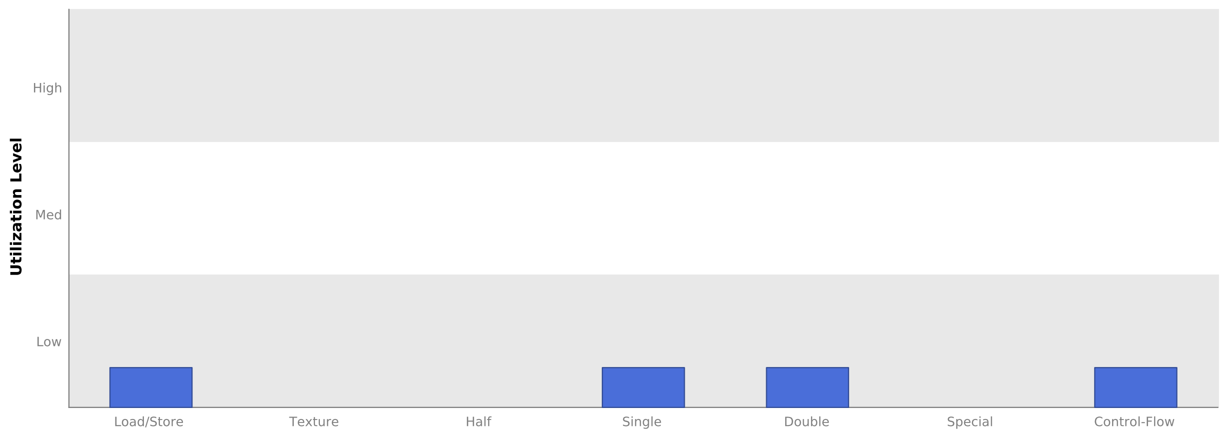
4. Compute Resources

GPU compute resources limit the performance of a kernel when those resources are insufficient or poorly utilized.

4.1. Function Unit Utilization

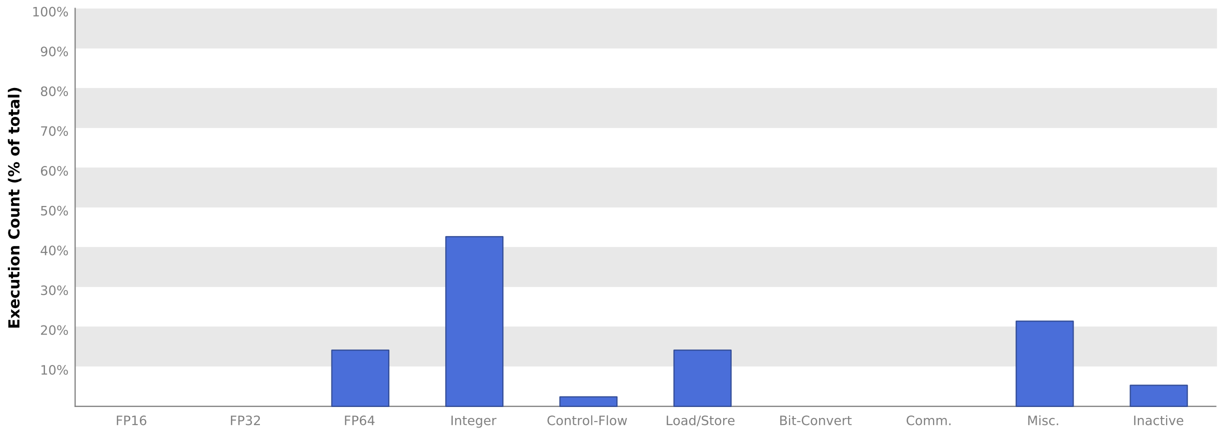
Different types of instructions are executed on different function units within each SM. Performance can be limited if a function unit is over-used by the instructions executed by the kernel. The following results show that the kernel's performance is not limited by overuse of any function unit.

Load/Store - Load and store instructions for shared and constant memory.
Texture - Load and store instructions for local, global, and texture memory.
Half - Half-precision floating-point arithmetic instructions.
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.



4.2. 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.



4.3. 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.