Analysis Report

matmult_gpu5Kernel(int, int, int, double*, double*, double*)

Duration	1.1295 s (1,129,504,552 ns)
Grid Size	[625,625,1]
Block Size	[8,8,1]
Registers/Thread	40
Shared Memory/Block	1 KiB
Shared Memory Requested	96 KiB
Shared Memory Executed	96 KiB
Shared Memory Bank Size	4 B

[0] Tesla V100-SXM2-32GB

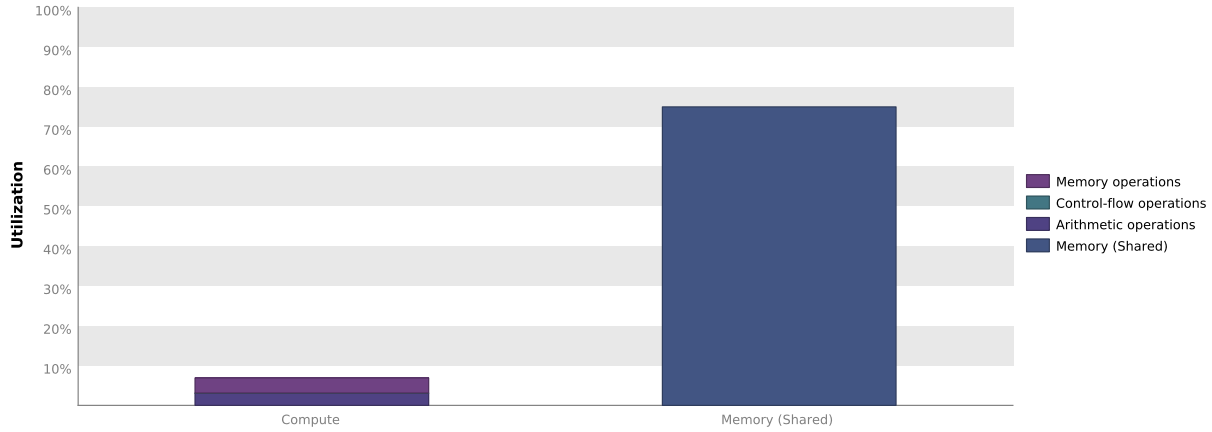
GPU UUID	GPU-c2fbdc1c-622a-33f0-430e-76df65ca5880
Compute Capability	7.0
Max. Threads per Block	1024
Max. Threads per Multiprocessor	2048
Max. Shared Memory per Block	48 KiB
Max. Shared Memory per Multiprocessor	96 KiB
Max. Registers per Block	65536
Max. Registers per Multiprocessor	65536
Max. Grid Dimensions	[2147483647, 65535, 65535]
Max. Block Dimensions	[1024, 1024, 64]
Max. Warps per Multiprocessor	64
Max. Blocks per Multiprocessor	32
Half Precision FLOP/s	31.334 TeraFLOP/s
Single Precision FLOP/s	15.667 TeraFLOP/s
Double Precision FLOP/s	7.834 TeraFLOP/s
Number of Multiprocessors	80
Multiprocessor Clock Rate	1.53 GHz
Concurrent Kernel	true
Max IPC	4
Threads per Warp	32
Global Memory Bandwidth	898.048 GB/s
Global Memory Size	31.719 GiB
Constant Memory Size	64 KiB
L2 Cache Size	6 MiB
Memcpy Engines	5
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 "matmult_gpu5Kernel" 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-SXM2-32GB" 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 Shared 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 shared memory.

2.1. Shared Memory Alignment and Access Pattern

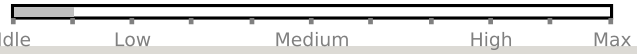
Memory bandwidth is used most efficiently when each shared memory load and store has proper alignment and access pattern.

Optimization: Select each entry below to open the source code to a shared 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	35158037478	9,301.523 GB/s	
Shared Stores	7944294034	2,101.768 GB/s	
Shared Total	43102331512	11,403.291 GB/s	
L2 Cache			
Reads	7472842953	494.26 GB/s	
Writes	14028019	927.824 MB/s	
Total	7486870972	495.188 GB/s	
Unified Cache			
Local Loads	2681748	177.373 MB/s	
Local Stores	4229120	279.717 MB/s	
Global Loads	7469327257	494.027 GB/s	
Global Stores	9485752	627.395 MB/s	
Texture Reads	18941765123	5,011.294 GB/s	
Unified Total	26427489000	5,506.406 GB/s	
Device Memory			
Reads	3935107005	260.271 GB/s	
Writes	13444723	889.245 MB/s	
Total	3948551728	261.16 GB/s	
System Memory			
[PCIe configuration: Gen3 x16, 8 Gbit/s]			
Reads	30773	2.035 MB/s	
Writes	30213	1.998 MB/s	

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. The results below indicate that occupancy can be improved by reducing the number of registers used by the kernel.

3.1. GPU Utilization May Be Limited By Register Usage

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.

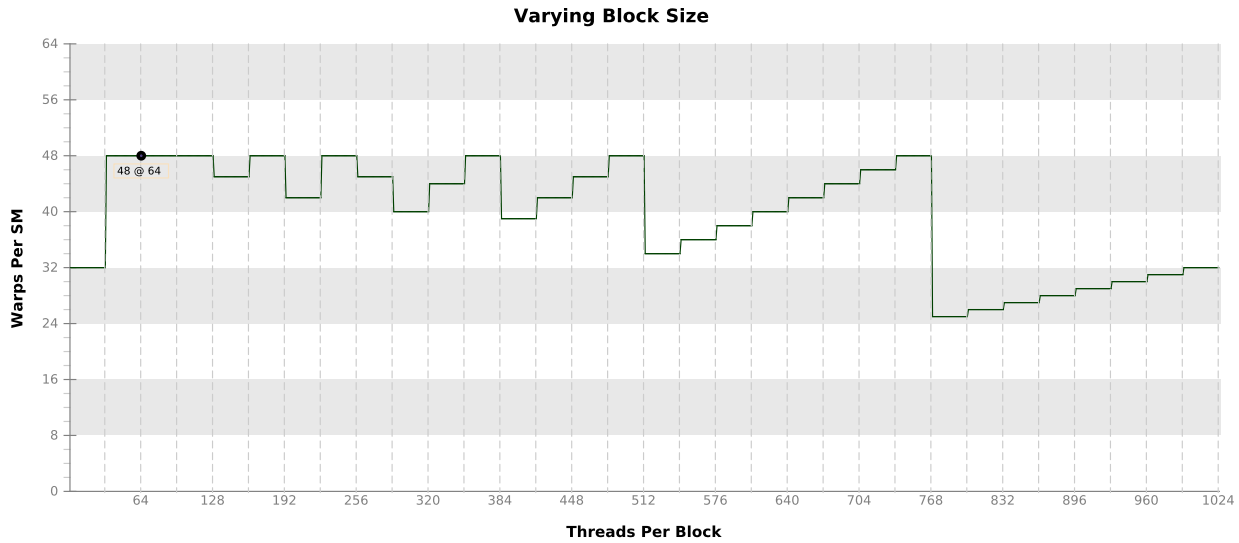
The kernel uses 40 registers for each thread (2560 registers for each block). This register usage is likely preventing the kernel from fully utilizing the GPU. Device "Tesla V100-SXM2-32GB" provides up to 65536 registers for each block. Because the kernel uses 2560 registers for each block each SM is limited to simultaneously executing 24 blocks (48 warps). Chart "Varying Register Count" below shows how changing register usage will change the number of blocks that can execute on each SM.

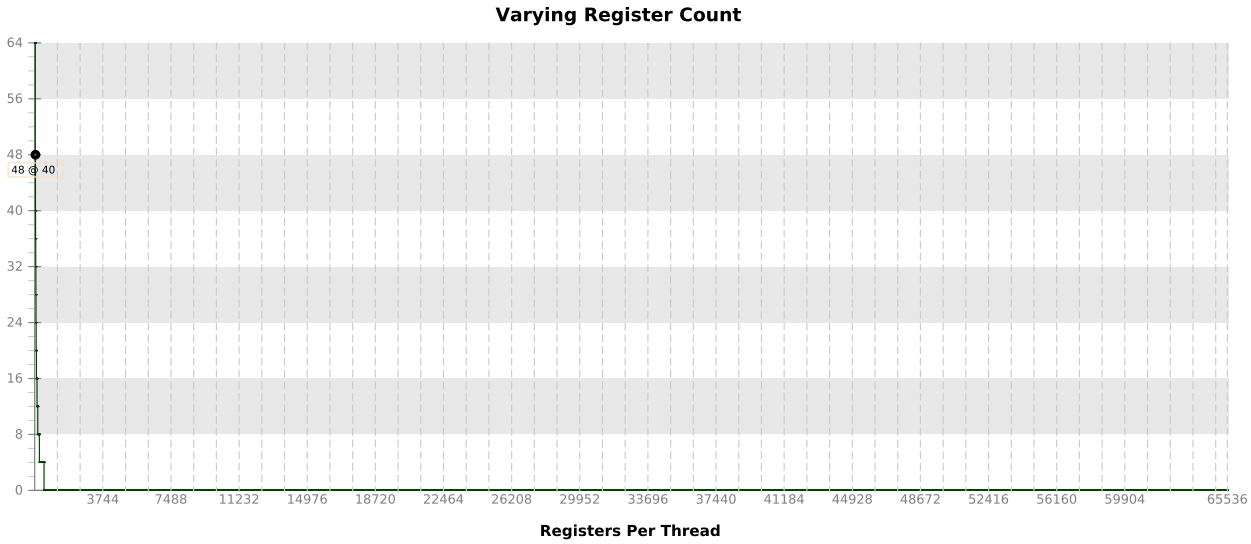
Optimization: Use the -maxrregcount flag or the __launch_bounds__ qualifier to decrease the number of registers used by each thread. This will increase the number of blocks that can execute on each SM. On devices with Compute Capability 5.2 turning global cache off can increase the occupancy limited by register usage.

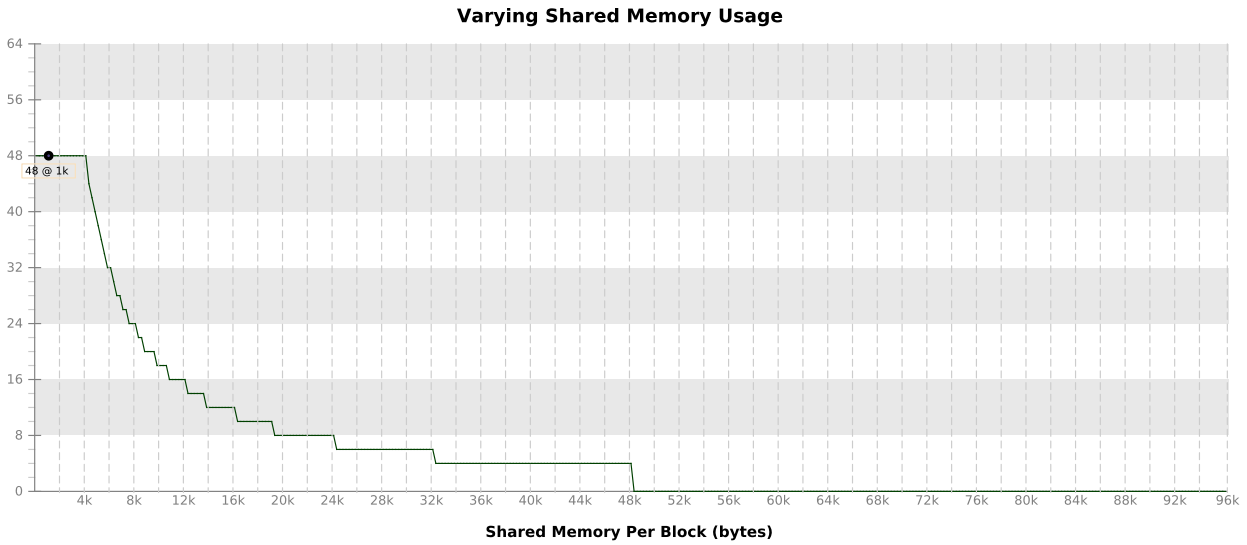
Variable	Achieved	Theoretical	Device Limit	Grid Size: [625,625,1] (390625 blocks) Block Size: [8,8,1] (64 th
Occupancy Per SM				
Active Blocks		24	32	0 3 6 9 12 15 18 21 24 27 30 32
Active Warps	47.1	48	64	0 7 14 21 28 35 42 49 56 64
Active Threads		1536	2048	0 256 512 768 1024 1280 1536 1792 2048
Occupancy	73.6%	75%	100%	0% 25% 50% 75% 100%
Warps				
Threads/Block		64	1024	0 128 256 384 512 640 768 896 1024
Warps/Block		2	32	0 3 6 9 12 15 18 21 24 27 30 32
Block Limit		32	32	0 3 6 9 12 15 18 21 24 27 30 32
Registers				
Registers/Thread		40	65536	0 8192 16384 24576 32768 40960 49152 57344 65536
Registers/Block		2560	65536	0 16k 32k 48k 64k
Block Limit		24	32	0 3 6 9 12 15 18 21 24 27 30 32
Shared Memory				
Shared Memory/Block		1024	98304	0 32k 64k 96k
Block Limit		96	32	0 3 6 9 12 15 18 21 24 27 30 32

3.2. Occupancy Charts

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

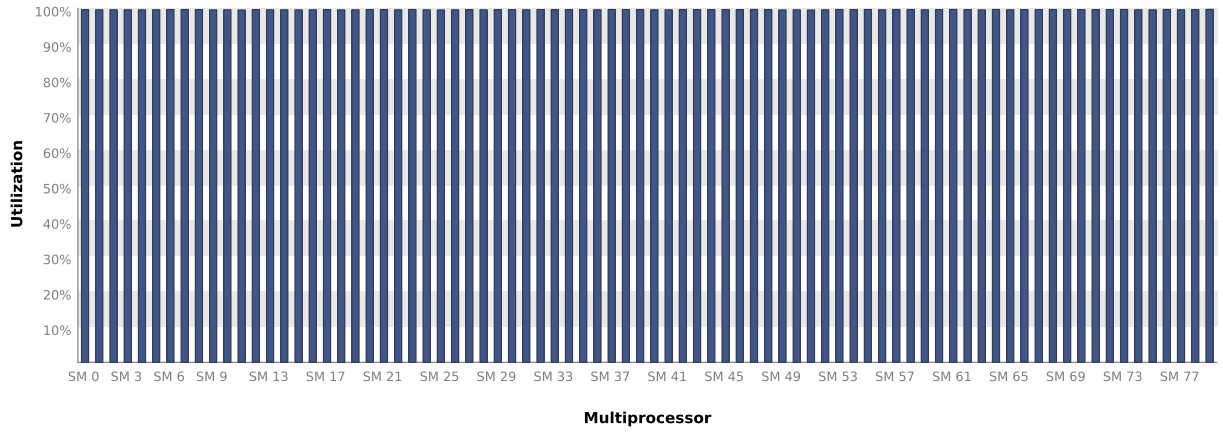

Varying Block Size
Warps Per SM
64
56
48
40
32
24
16
8
0
48 @ 64
64
128
192
256
320
384
448
512
576
640
704
768
832
896
960
1024
Threads Per Block


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



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.



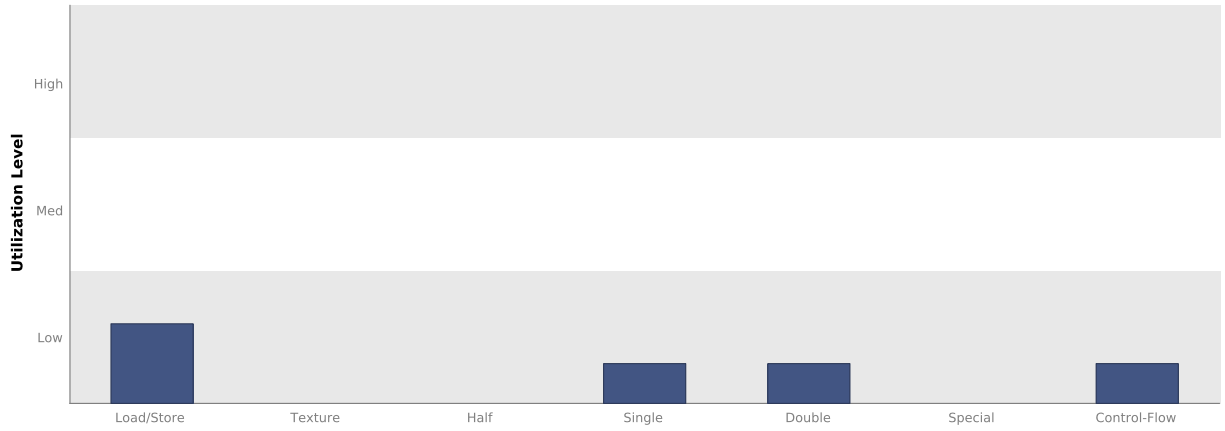
4. Compute Resources

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

4.1. Function Unit Utilization

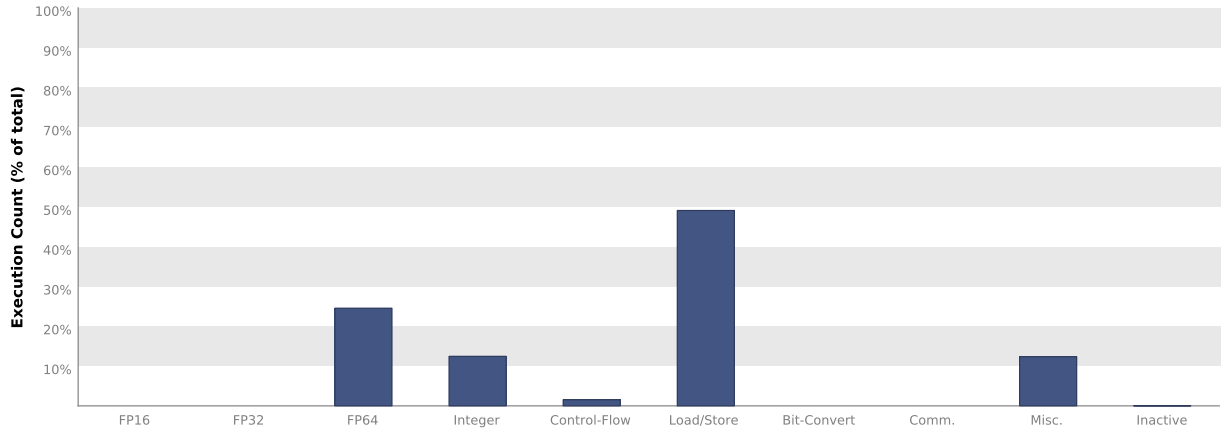
Different types of instructions are executed on different function units within each SM. Performance can be limited if a function unit is over-used by the instructions executed by the kernel. The following results show that the kernel's performance is not limited by overuse of any function unit.

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.