

# Analysis Report

## volta\_dgemm\_128x64\_nn

Duration	73.5511 ms (73,551,095 ns)
Grid Size	[ 40,79,1 ]
Block Size	[ 128,1,1 ]
Registers/Thread	234
Shared Memory/Block	24.5 KiB
Shared Memory Requested	96 KiB
Shared Memory Executed	96 KiB
Shared Memory Bank Size	4 B

### [0] Tesla V100-SXM2-32GB

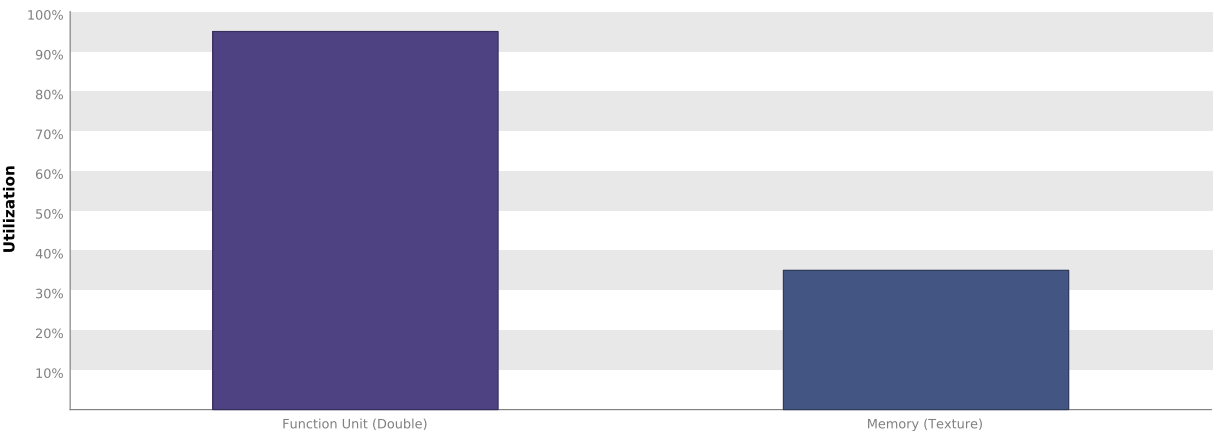
GPU UUID	GPU-c2fbdc1c-622a-33f0-430e-76df65ca5880
Compute Capability	7.0
Max. Threads per Block	1024
Max. Threads per Multiprocessor	2048
Max. Shared Memory per Block	48 KiB
Max. Shared Memory per Multiprocessor	96 KiB
Max. Registers per Block	65536
Max. Registers per Multiprocessor	65536
Max. Grid Dimensions	[ 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 "volta\_dgemm\_128x64\_nn" 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 "Tesla V100-SXM2-32GB" 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 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.

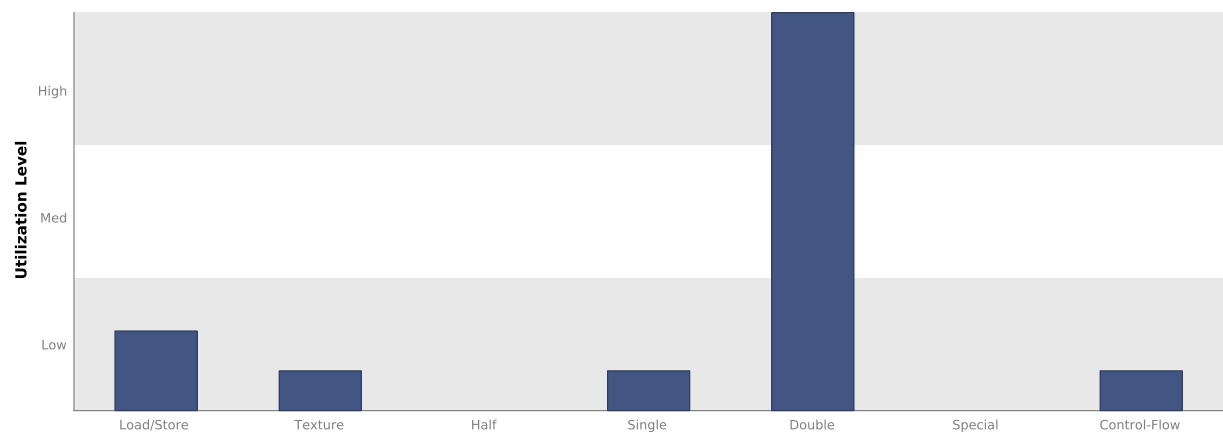
*Examine portions of the kernel that have high execution counts and inactive or predicated threads to identify optimization opportunities.*

Cuda Fuctions :
volta_dgemm_128x64_nn
Maximum instruction execution count in assembly: 7900000
Average instruction execution count in assembly: 4457945
Instructions executed for the kernel: 4948319840
Thread instructions executed for the kernel: 154687308800
Non-predicated thread instructions executed for the kernel: 156410586720
Warp non-predicated execution efficiency of the kernel: 98.8%
Warp execution efficiency of the kernel: 97.7%

### 2.2. GPU Utilization Is Limited By Function Unit Usage

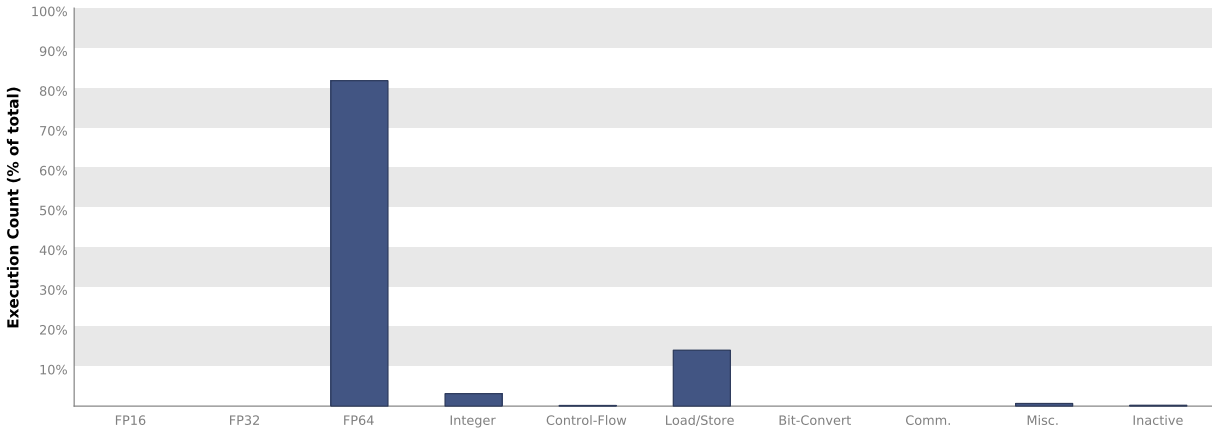
Different types of instructions are executed on different function units within each SM. Performance can be limited if a function unit is over-used by the instructions executed by the kernel. The following results show that the kernel's performance is potentially limited by overuse of the following function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

- 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.



### 2.3. 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.



### 2.4. 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.









### 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.

#### 3.1. 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	
Shared Memory			
Shared Loads	1014387073	3,884.4 GB/s	
Shared Stores	256052567	980.504 GB/s	
Shared Total	1270439640	4,864.904 GB/s	
L2 Cache			
Reads	740675831	709.069 GB/s	
Writes	6273889	6.006 GB/s	
Total	746949720	715.075 GB/s	
Unified Cache			
Local Loads	0	0 B/s	
Local Stores	0	0 B/s	
Global Loads	683781428	654.602 GB/s	
Global Stores	6250000	5.983 GB/s	
Texture Reads	1238105793	4,741.088 GB/s	
Unified Total	1928137221	5,401.674 GB/s	
Device Memory			
Reads	303977534	291.006 GB/s	
Writes	6316933	6.047 GB/s	
Total	310294467	297.053 GB/s	
System Memory			
[ PCIe configuration: Gen3 x16, 8 Gbit/s ]			
Reads	0	0 B/s	
Writes	5	4.786 kB/s	

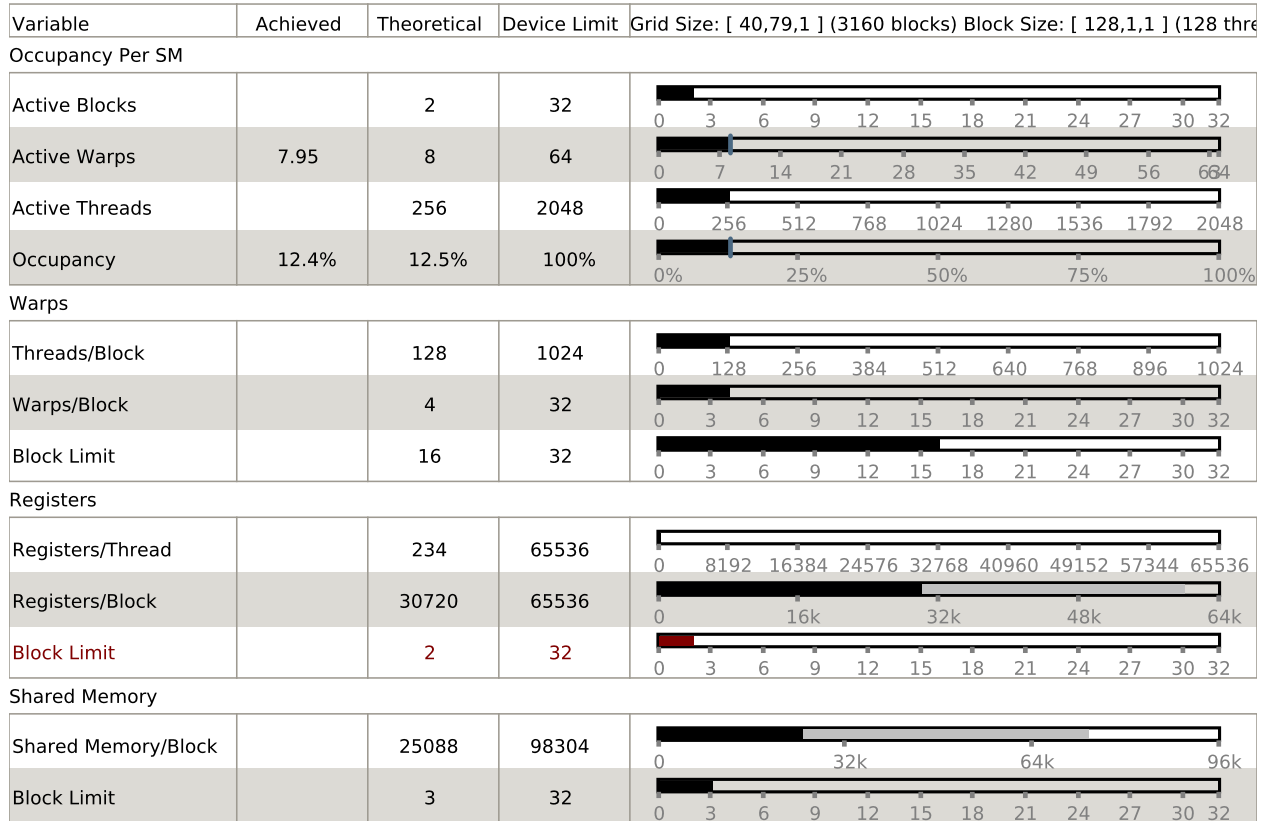
## 4. Instruction and Memory Latency

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 reducing the number of registers used by the kernel.

### 4.1. GPU Utilization Is Limited By Register Usage

The kernel uses 234 registers for each thread (29952 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 29952 registers for each block each SM is limited to simultaneously executing 2 blocks (8 warps). Chart "Varying Register Count" below shows how changing register usage will change the number of blocks that can execute on each SM.

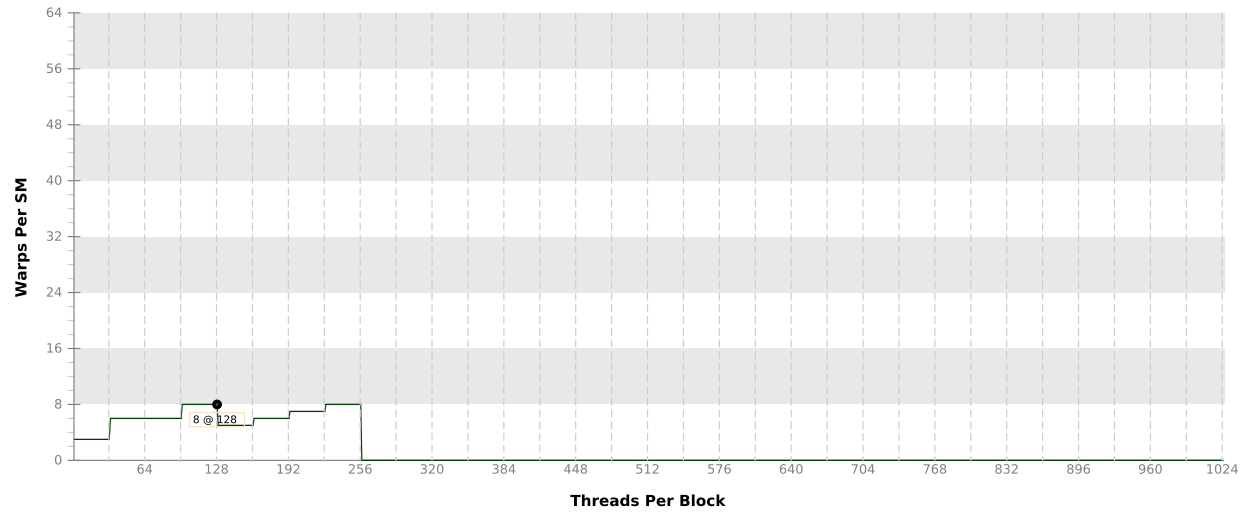
*Optimization: Use the `-maxrregcount` flag or the `__launch_bounds__` qualifier to decrease the number of registers used by each thread. This will increase the number of blocks that can execute on each SM. On devices with Compute Capability 5.2 turning global cache off can increase the occupancy limited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



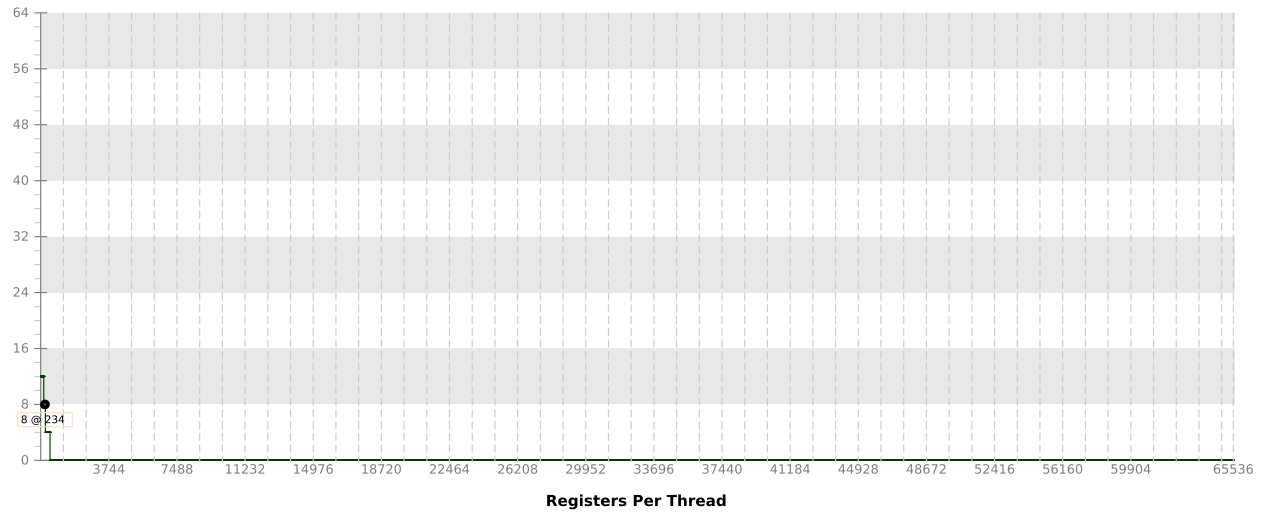
### 4.2. Occupancy Charts

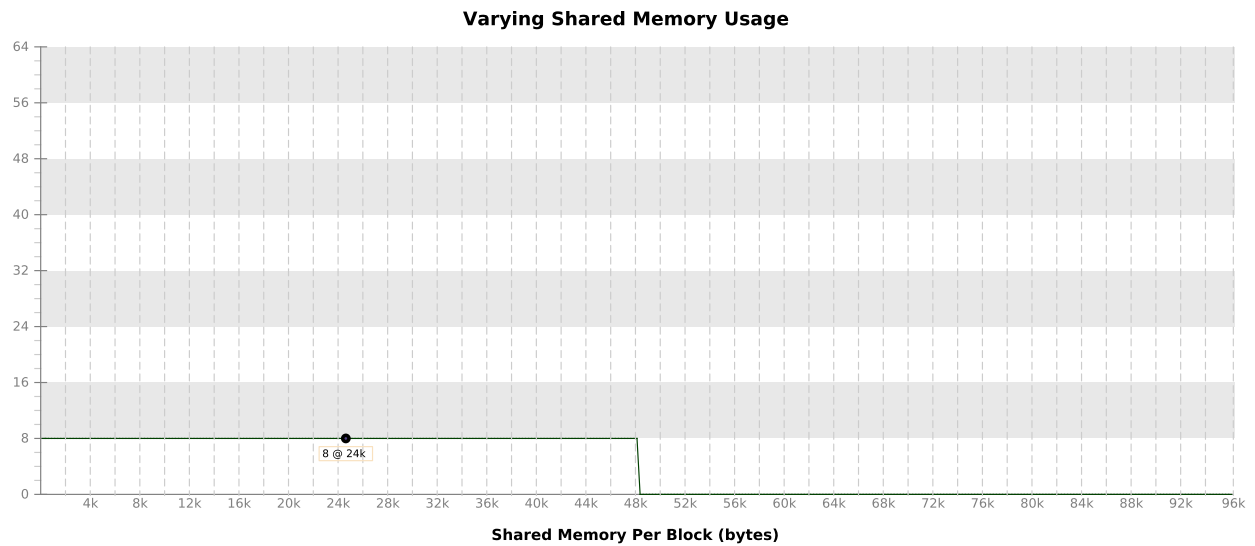
The following charts show how varying different components of the kernel will impact theoretical occupancy.

**Varying Block Size**



**Varying Register Count**





### 4.3. 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.

