Analysis Report

spmv_kernel(float const *, int const *, int const *, float const *, int, float*)

Duration	1.011 ms (1,010,763 ns)
Grid Size	[2016,1,1]
Block Size	[256,1,1]
Registers/Thread	21
Shared Memory/Block	1 KiB
Shared Memory Requested	48 KiB
Shared Memory Executed	48 KiB
Shared Memory Bank Size	4 B

[3] Tesla K40c

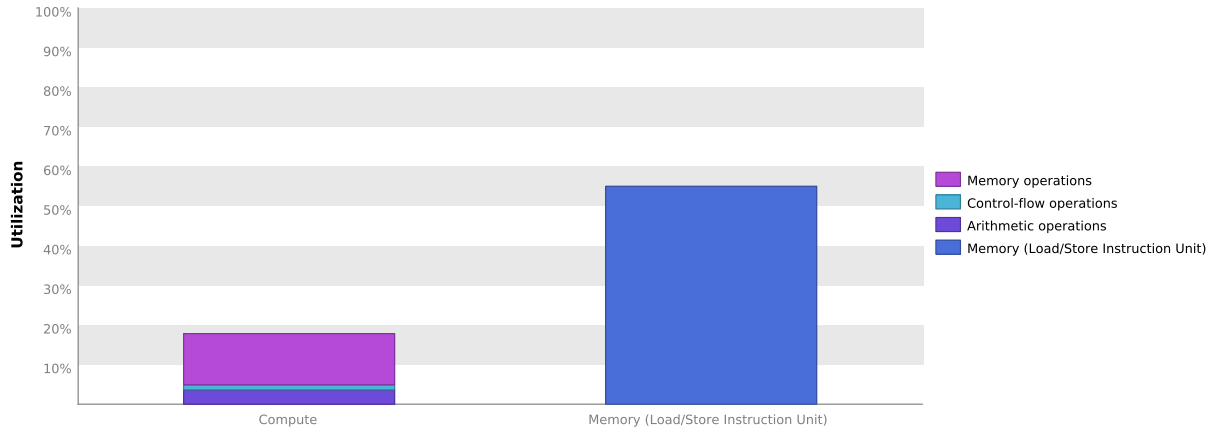
GPU UUID	GPU-325599d1-cb86-f7c9-1c97-6aaa2eb17839
Compute Capability	3.5
Max. Threads per Block	1024
Max. Threads per Multiprocessor	2048
Max. Shared Memory per Block	48 KiB
Max. Shared Memory per Multiprocessor	48 KiB
Max. Registers per Block	65536
Max. Registers per Multiprocessor	65536
Max. Grid Dimensions	[2147483647, 65535, 65535]
Max. Block Dimensions	[1024, 1024, 64]
Max. Warps per Multiprocessor	64
Max. Blocks per Multiprocessor	16
Single Precision FLOP/s	4.291 TeraFLOP/s
Double Precision FLOP/s	1.43 TeraFLOP/s
Number of Multiprocessors	15
Multiprocessor Clock Rate	745 MHz
Concurrent Kernel	true
Max IPC	7
Threads per Warp	32
Global Memory Bandwidth	288.384 GB/s
Global Memory Size	11.173 GiB
Constant Memory Size	64 KiB
L2 Cache Size	1.5 MiB
Memcpy Engines	2
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 "spmv_kernel" 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 K40c". 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. The results below indicate that the GPU does not have enough work because instruction execution is stalling excessively.

2.1. Instruction Latencies May Be Limiting Performance

Instruction stall reasons indicate the condition that prevents warps from executing on any given cycle. The following chart shows the break-down of stalls reasons averaged over the entire execution of the kernel. The kernel has good theoretical and achieved occupancy indicating that there are likely sufficient warps executing on each SM. Since occupancy is not an issue it is likely that performance is limited by the instruction stall reasons described below.

Pipeline Busy - The compute resource(s) required by the instruction is not yet available.

Not Selected - Warp was ready to issue, but some other warp issued instead. You may be able to sacrifice occupancy without impacting latency hiding and doing so may help improve cache hit rates.

Execution Dependency - An input required by the instruction is not yet available. Execution dependency stalls can potentially be reduced by increasing instruction-level parallelism.

Memory Dependency - A load/store cannot be made because the required resources are not available or are fully utilized, or too many requests of a given type are outstanding. Data request stalls can potentially be reduced by optimizing memory alignment and access patterns.

Synchronization - The warp is blocked at a __syncthreads() call.

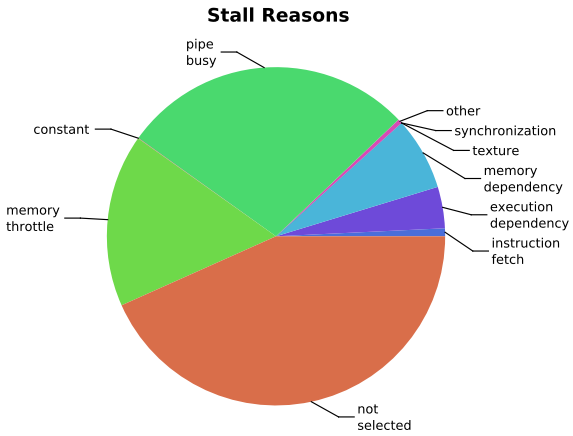
Memory Throttle - Large number of pending memory operations prevent further forward progress. These can be reduced by combining several memory transactions into one.

Texture - The texture sub-system is fully utilized or has too many outstanding requests.

Instruction Fetch - The next assembly instruction has not yet been fetched.

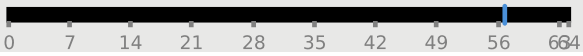
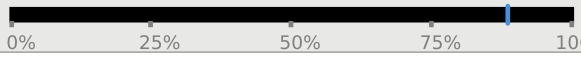
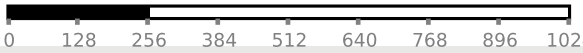
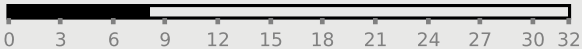
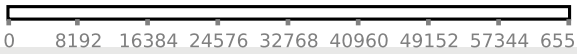
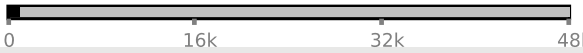
Constant - A constant load is blocked due to a miss in the constants cache.

Optimization: Resolve the primary stall issue; not selected.



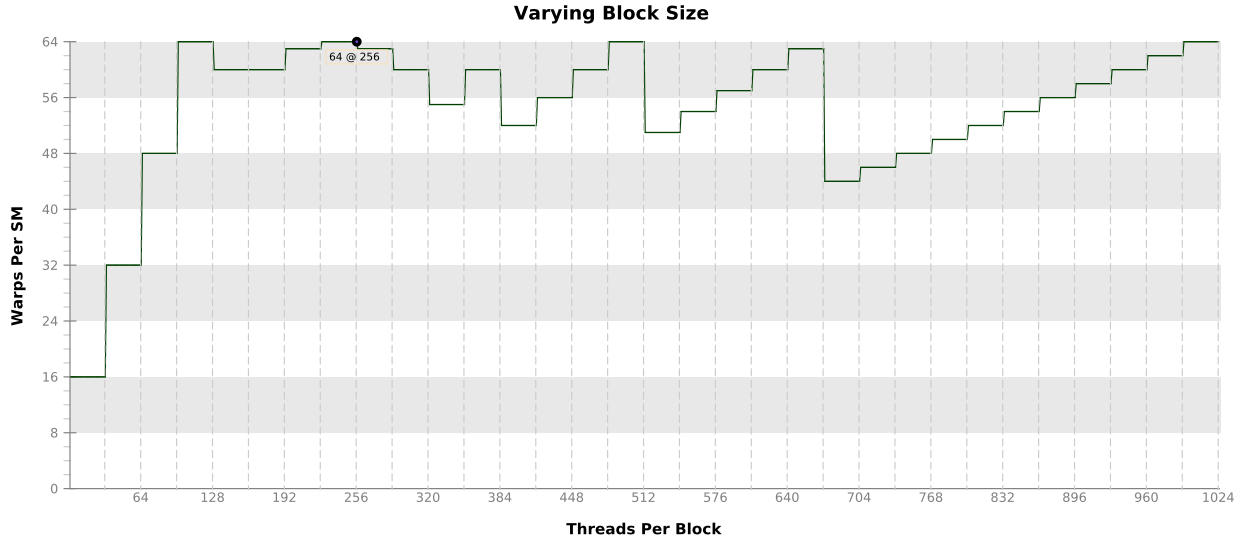
2.2. Occupancy Is Not Limiting Kernel Performance

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

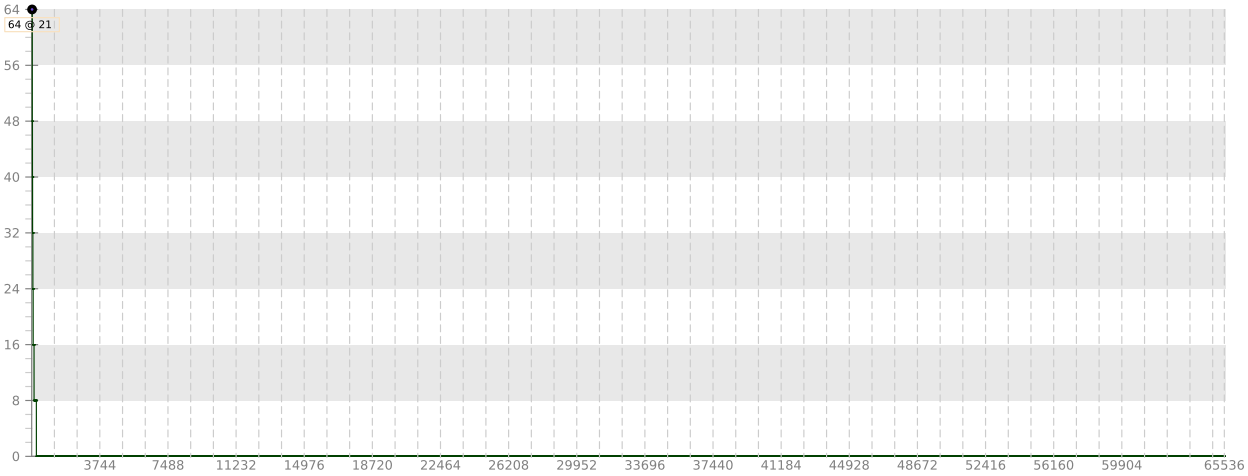
Variable	Achieved	Theoretical	Device Limit	Grid Size: [2016,1,1] (2016 blocks) Block Size: [256,1,1] (256 th
Occupancy Per SM				
Active Blocks		8	16	
Active Warps	56.5	64	64	
Active Threads		2048	2048	
Occupancy	88.3%	100%	100%	
Warps				
Threads/Block		256	1024	
Warps/Block		8	32	
Block Limit		8	16	
Registers				
Registers/Thread		21	65536	
Registers/Block		6144	65536	
Block Limit		10	16	
Shared Memory				
Shared Memory/Block		1024	49152	
Block Limit		48	16	

2.3. Occupancy Charts

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

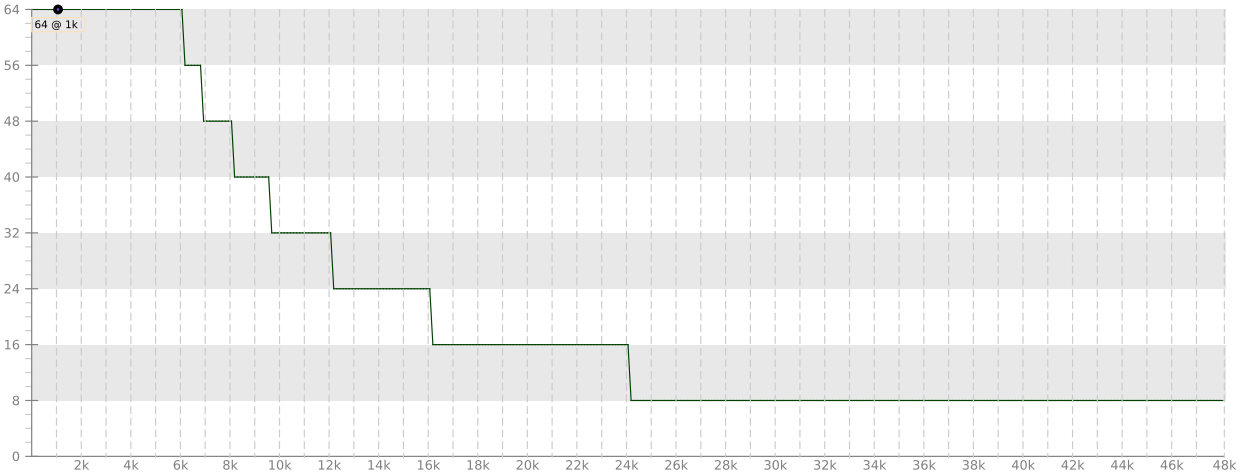


Varying Register Count



Registers Per Thread

Varying Shared Memory Usage



Shared Memory Per Block (bytes)

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. 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 :

spmv_kernel(float const *, int const *, int const *, float const *, int, float*)

Maximum instruction execution count in assembly: 81048
Average instruction execution count in assembly: 24507
Instructions executed for the kernel: 3087972
Thread instructions executed for the kernel: 78343111
Non-predicated thread instructions executed for the kernel: 75353441
Warp non-predicated execution efficiency of the kernel: 76.3%
Warp execution efficiency of the kernel: 79.3%

Source files :

/home/lin32/Development/projects/DataPlacement/PORPLE/OrgAndOptBenchmarks/spmv/spmv.cu

3.2. 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 77% 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 79.9%.

Optimization: Reduce the amount of intra-warp divergence and predication in the kernel.

3.3. 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.

/home/lin32/Development/projects/DataPlacement/PORPLE/OrgAndOptBenchmarks/spmv/spmv.cu

Line 169	Divergence = 0% [0 divergent executions out of 16128 total executions]
Line 174	Divergence = 9.2% [7469 divergent executions out of 81048 total executions]
Line 174	Divergence = 0% [0 divergent executions out of 16128 total executions]
Line 174	Divergence = 96.8% [15613 divergent executions out of 16128 total executions]
Line 174	Divergence = 0% [0 divergent executions out of 16128 total executions]
Line 174	Divergence = 0% [0 divergent executions out of 15750 total executions]

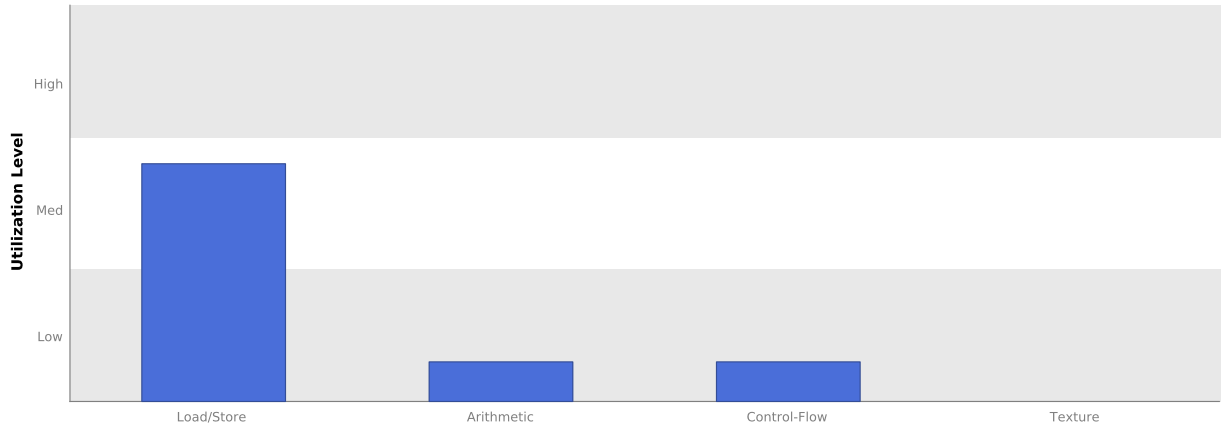
/home/lin32/Development/projects/DataPlacement/PORPLE/OrgAndOptBenchmarks/spmv/spmv.cu

Line 185	Divergence = 100% [16128 divergent executions out of 16128 total executions]
Line 193	Divergence = 0% [0 divergent executions out of 16128 total executions]

3.4. Function Unit Utilization

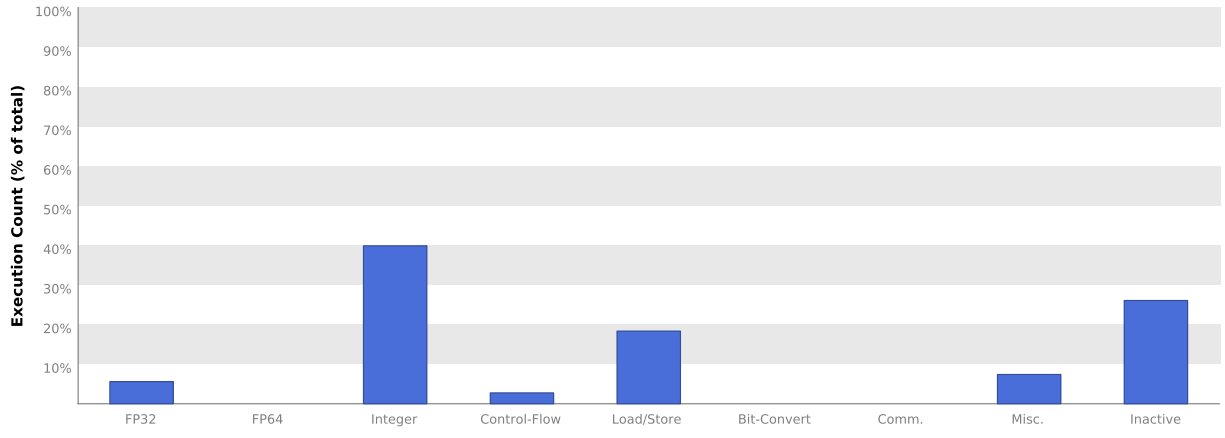
Different types of instructions are executed on different function units within each SM. Performance can be limited if a function unit is over-used by the instructions executed by the kernel. The following results show that the kernel's performance is not limited by overuse of any function unit.

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.



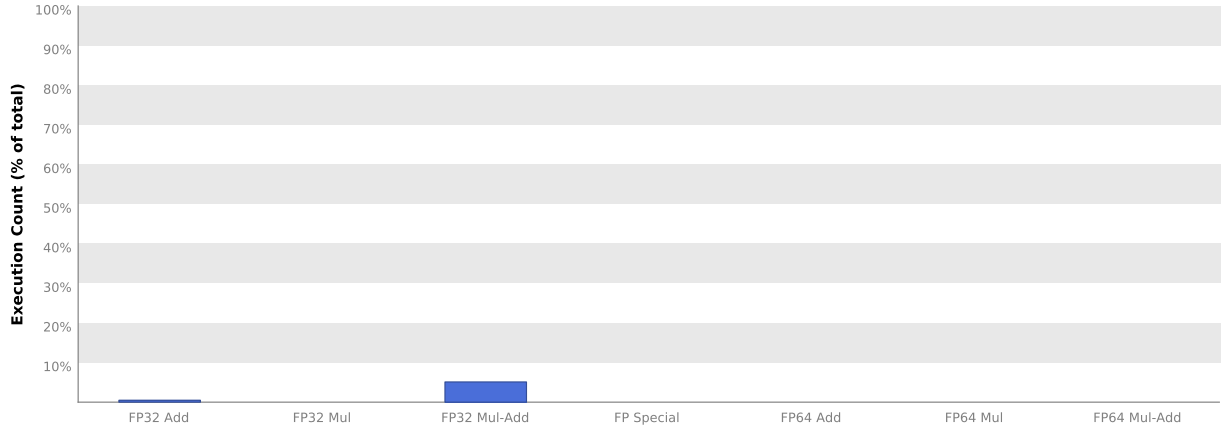
3.5. Instruction Execution Counts

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.



3.6. 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.

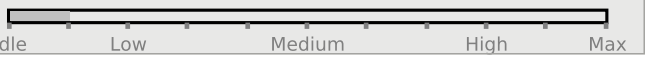
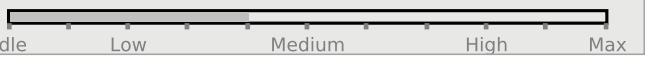
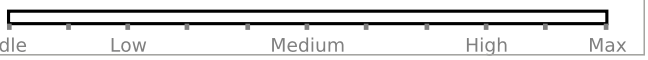
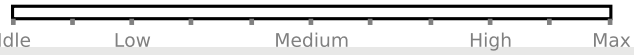


4. Memory Bandwidth

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.

4.1. 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	
L1/Shared Memory			
Local Loads	0	0 B/s	
Local Stores	0	0 B/s	
Shared Loads	177408	45.497 GB/s	
Shared Stores	96768	24.817 GB/s	
Global Loads	4611846	208.539 GB/s	
Global Stores	16128	517.013 MB/s	
Atomic	0	0 B/s	
L1/Shared Total	4902150	279.37 GB/s	
L2 Cache			
L1 Reads	6505289	208.539 GB/s	
L1 Writes	16128	517.013 MB/s	
Texture Reads	0	0 B/s	
Noncoherent Reads	0	0 B/s	
Atomic	0	0 B/s	
Total	6521417	209.056 GB/s	
Texture Cache			
Reads	0	0 B/s	
Device Memory			
Reads	1862390	59.702 GB/s	
Writes	3412	109.378 MB/s	
Total	1865802	59.812 GB/s	
ECC Overhead	554840	17.786 GB/s	
System Memory			
[PCIe configuration: Gen3 x16, 8 Gbit/s]			
Reads	0	0 B/s	
Writes	6	192.341 kB/s