Analysis Report

jacobiIteration(double*, double*, double*, int)

|	
Duration	1.48938 s (1,489,380,692 ns)
Grid Size	[1,1,1]
Block Size	[1,1,1]
Registers/Thread	31
Shared Memory/Block	0 B
Shared Memory Executed	0 B
Shared Memory Bank Size	4 B

[0] Tesla V100-SXM2-32GB	
GPU UUID	GPU-716a9097-dd3b-c568-7482-6fb712b41563
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 "jacobiIteration" 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 V100-SXM2-32GB". 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.
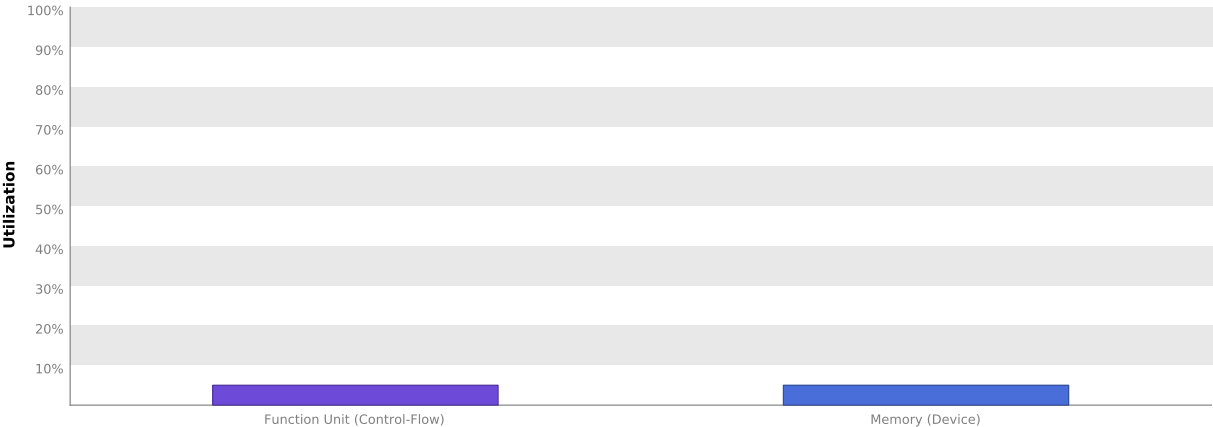
2. Instruction and Memory Latency

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 the kernel does not execute enough blocks.

2.1. Grid Size Too Small To Hide Compute And Memory Latency

The kernel does not execute enough blocks to hide memory and operation latency. Typically the kernel grid size must be large enough to fill the GPU with multiple "waves" of blocks. Based on theoretical occupancy, device "Tesla V100-SXM2-32GB" can simultaneously execute 32 blocks on each of the 80 SMs, so the kernel may need to execute a multiple of 2560 blocks to hide the compute and memory latency. If the kernel is executing concurrently with other kernels then fewer blocks will be required because the kernel is sharing the SMs with those kernels.

Optimization: Increase the number of blocks executed by the kernel.

2.2. GPU Utilization May Be Limited By Block Size

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 has a block size of 1 thread. This block size is likely preventing the kernel from fully utilizing the GPU. Device "Tesla V100-SXM2-32GB" can simultaneously execute up to 32 blocks on each SM. Because each block uses 1 warp to execute the block's 1 thread, the kernel is using only 32 warps on each SM. Chart "Varying Block Size" below shows how changing the block size will change the number of warps that can execute on each SM.

Optimization: Increase the number of threads in each block to increase the number of warps that can execute on each SM.

Variable	Achieved	Theoretical	Device Limit	Grid Size: [1,1,1] (1 block) Block Size: [1,1,1] (1 thread)
Occupancy Per SM				
Active Blocks		32	32	
Active Warps	1	32	64	
Active Threads		1024	2048	
Occupancy	1.6%	50%	100%	
Warps				
Threads/Block		1	1024	
Warps/Block		1	32	
Block Limit		64	32	
Registers				
Registers/Thread		31	65536	
Registers/Block		1024	65536	
Block Limit		64	32	
Shared Memory				
Shared Memory/Block		0	98304	
Block Limit		0	32	

2.3. Occupancy Charts

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

Varying Block Size

Varying Register Count

Varying Shared Memory Usage

Shared Memory Per Block (bytes)
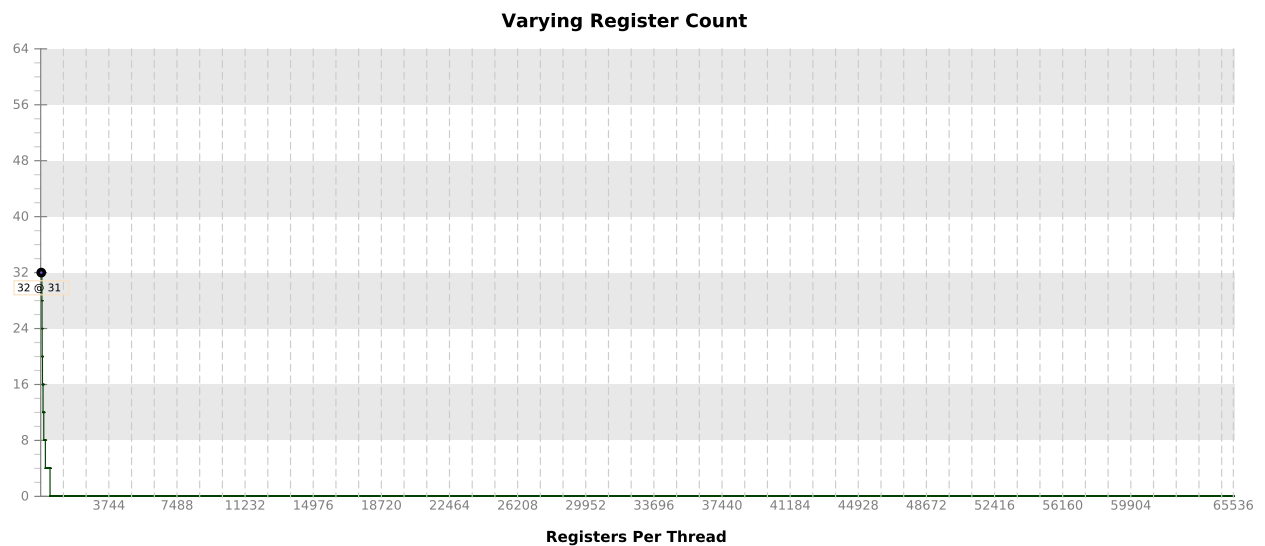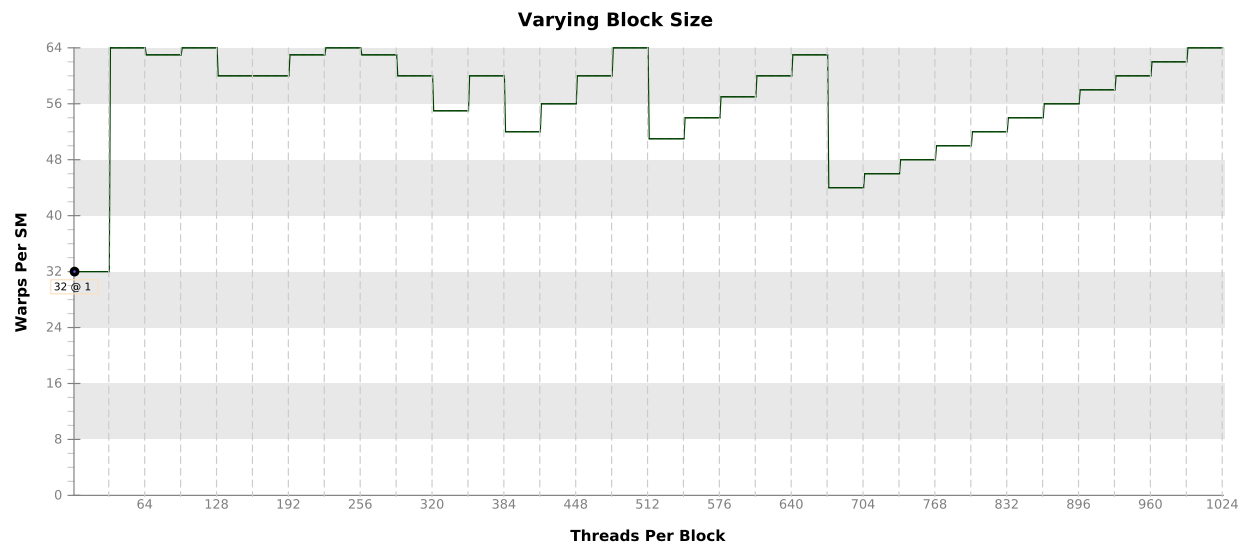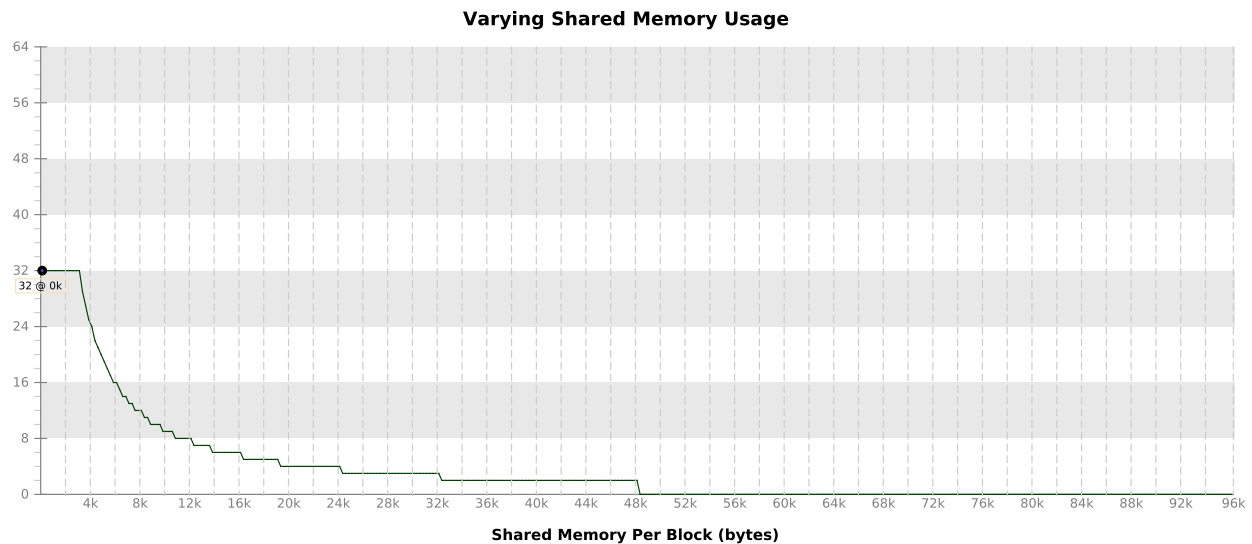
2.4. Multiprocessor Utilization

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.

Multiprocessor

2.5. Kernel Profile - PC Sampling

The Kernel Profile - PC Sampling gives the number of samples for each source and assembly line with various stall reasons. The samples are collected at a period of 32 [2^5] cycles. You can change the period under Settings->Analysis tab. The allowed values are from 5 to 31. Increasing the period would reduce the number of samples collected.
 Using this information you can pinpoint portions of your kernel that are introducing latencies and the reason for the latency.
Samples are taken in round robin order for all active warps at a fixed number of cycles regardless of whether the warp is issuing an instruction or not.
 Instruction Issued - Warp was issued
 Instruction Fetch - The next assembly instruction has not yet been fetched.
 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.

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

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

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

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

 Memory Throttle - Large number of pending memory operations prevent further forward progress. These can be reduced by combining several memory transactions into one.
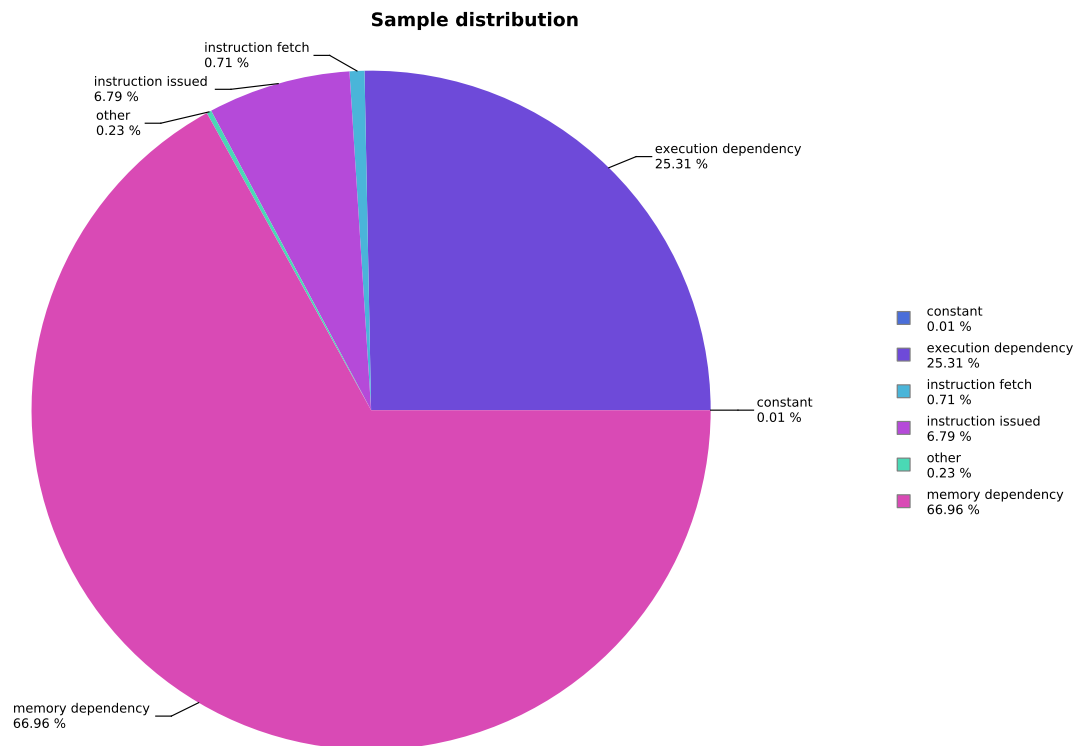
 Not Selected - Warp was ready to issue, but some other warp issued instead. You may be able to sacrifice occupancy without impacting latency hiding and doing so may help improve cache hit rates.

 Other - The warp is blocked for an uncommon reason.

 Sleeping -The warp is blocked, yielded or sleeping.

Examine portions of the kernel that have high number of samples to know where the maximum time was spent and observe the latency reasons for those samples to identify optimization opportunities.

Cuda Functions	Sample Count	% of Kernel Samples
jacobiIteration(double*, double*, double*, int)	7002909	100.0

Sample distribution

instruction fetch
0.71 %

instruction issued
6.79 %

other
0.23 %

execution dependency
25.31 %

constant
0.01 %

memory dependency
66.96 %

constant
0.01 %

execution dependency
25.31 %

instruction fetch
0.71 %

instruction issued
6.79 %

other
0.23 %

memory dependency
66.96 %

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 :

jacobiIteration(double*, double*, double*, int)

Maximum instruction execution count in assembly: 262144
Average instruction execution count in assembly: 165565
Instructions executed for the kernel: 15232007
Thread instructions executed for the kernel: 15232007
Non-predicated thread instructions executed for the kernel: 15228932
Warp non-predicated execution efficiency of the kernel: 3.1%
Warp execution efficiency of the kernel: 3.1%

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 maximum warp execution efficiency is 3.1% because the number of threads per block is not a multiple of the warp size.

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

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 shared and constant memory.
Texture - Load and store instructions for local, global, and texture memory.
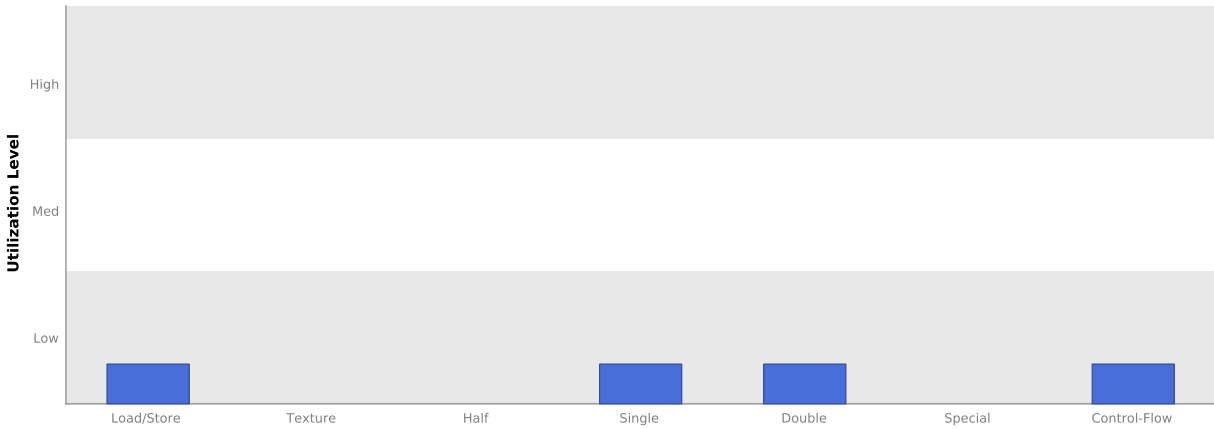Half - Half-precision floating-point arithmetic instructions.
Single - Single-precision integer and floating-point arithmetic instructions.
Double - Double-precision floating-point arithmetic instructions.
Special - Special arithmetic instructions such as sin, cos, popc, etc.
Control-Flow - Direct and indirect branches, jumps, and calls.

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.
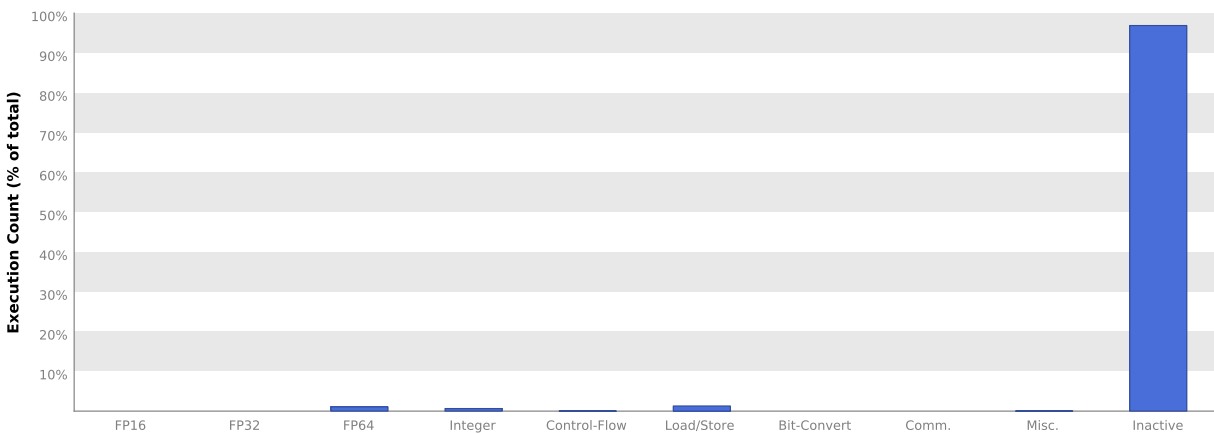
3.5. Floating-Point Operation Counts

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.

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	
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	528173	108.666 MB/s	
Writes	1102366	226.8 MB/s	
Total	1630539	335.466 MB/s	Idle — Low — Medium — High — Max
Unified Cache			
Local Loads	0	0 B/s	
Local Stores	0	0 B/s	
Global Loads	5242880	1.079 GB/s	
Global Stores	1048576	215.734 MB/s	
Texture Reads	5242880	4.315 GB/s	
Unified Total	11534336	5.609 GB/s	Idle — Low — Medium — High — Max
Device Memory			
Reads	528863	108.808 MB/s	
Writes	362019	74.482 MB/s	
Total	890882	183.29 MB/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	1.028 kB/s	Idle — Low — Medium — High — Max

4.2. Memory Statistics

The following chart shows a summary view of the memory hierarchy of the CUDA programming model. The green nodes in the diagram depict logical memory space whereas blue nodes depicts actual hardware unit on the chip. For the various caches the reported percentage number states the cache hit rate; that is the ratio of requests that could be served with data locally available to the cache over all requests made.

The links between the nodes in the diagram depict the data paths between the SMs to the memory spaces into the memory system. Different metrics are shown per data path. The data paths from the SMs to the memory spaces report the total number of memory instructions executed, it includes both read and write operations. The data path between memory spaces and "Unified Cache" or "Shared Memory" reports the total amount of memory requests made (read or write). All other data paths report the total amount of transferred memory in bytes.