Analysis Report

mxnet::op::forward_kernel(float*, float const *, float const *, int, int, int, int, int, int)

Duration	3.03194 ms (3,031,939 ns)
Grid Size	[1000,12,25]
Block Size	[16,16,1]
Registers/Thread	32
Shared Memory/Block	0 B
Shared Memory Executed	0 B
Shared Memory Bank Size	4 B

[0] TITAN V

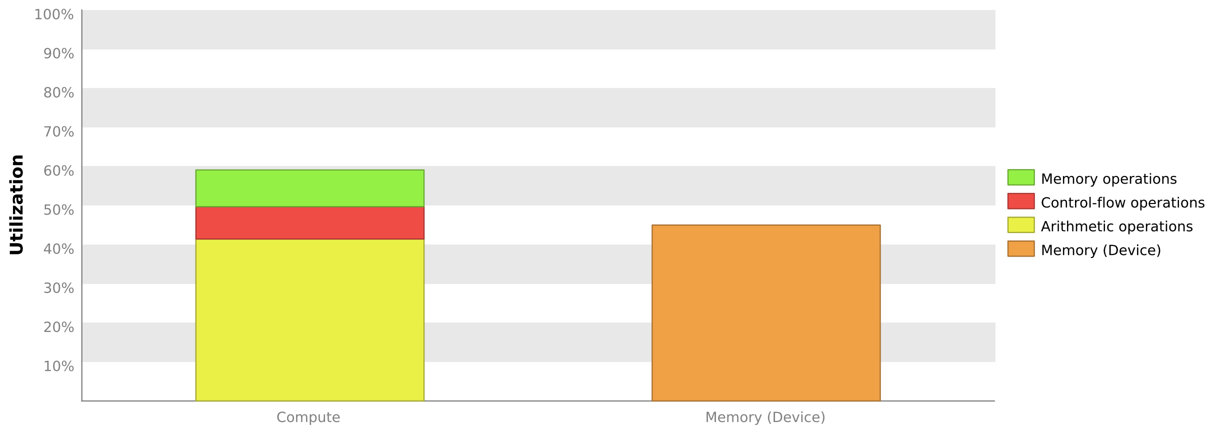
GPU UUID	GPU-9910c808-3335-47c7-980a-bf909aedc5aa
Compute Capability	7.0
Max. Threads per Block	1024
Max. Threads per Multiprocessor	2048
Max. Shared Memory per Block	48 KiB
Max. Shared Memory per Multiprocessor	96 KiB
Max. Registers per Block	65536
Max. Registers per Multiprocessor	65536
Max. Grid Dimensions	[2147483647, 65535, 65535]
Max. Block Dimensions	[1024, 1024, 64]
Max. Warps per Multiprocessor	64
Max. Blocks per Multiprocessor	32
Half Precision FLOP/s	29.798 TeraFLOP/s
Single Precision FLOP/s	14.899 TeraFLOP/s
Double Precision FLOP/s	7.45 TeraFLOP/s
Number of Multiprocessors	80
Multiprocessor Clock Rate	1.455 GHz
Concurrent Kernel	true
Max IPC	4
Threads per Warp	32
Global Memory Bandwidth	652.8 GB/s
Global Memory Size	11.755 GiB
Constant Memory Size	64 KiB
L2 Cache Size	4.5 MiB
Memcpy Engines	7
PCIe Generation	3
PCIe Link Rate	8 Gbit/s
PCIe Link Width	8

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 "mxnet::op::forward_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 "TITAN V". 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 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.

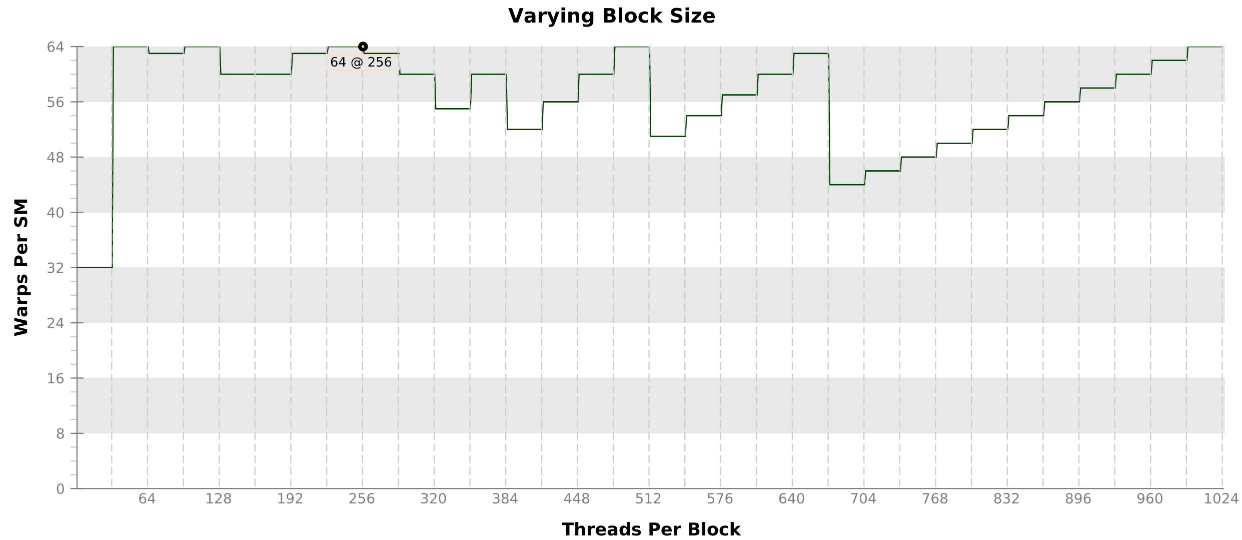
2.1. Occupancy Is Not Limiting Kernel Performance

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

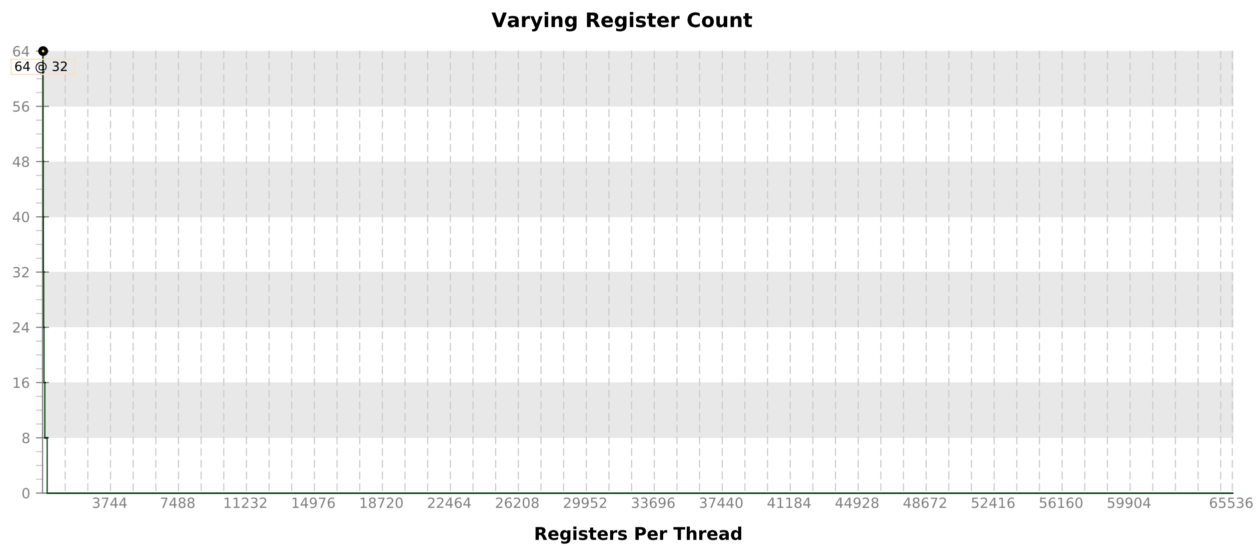
Variable	Achieved	Theoretical	Device Limit	Grid Size: [1000,12,25] (300000 blocks) Block Size: [1
Occupancy Per SM				
Active Blocks		8	32	0 4 8 12 16 20 24 28 32
Active Warps	52.28	64	64	0 9 18 27 36 45 54 63 64
Active Threads		2048	2048	0 512 1024 1536 2048
Occupancy	81.7%	100%	100%	0% 25% 50% 75% 100%
Warps				
Threads/Block		256	1024	0 256 512 768 1024
Warps/Block		8	32	0 4 8 12 16 20 24 28 32
Block Limit		8	32	0 4 8 12 16 20 24 28 32
Registers				
Registers/Thread		32	65536	0 16384 32768 49152 65536
Registers/Block		8192	65536	0 16k 32k 48k 64k
Block Limit		8	32	0 4 8 12 16 20 24 28 32
Shared Memory				
Shared Memory/Block		0	98304	0 32k 64k 96k
Block Limit		0	32	0 4 8 12 16 20 24 28 32

2.2. Occupancy Charts

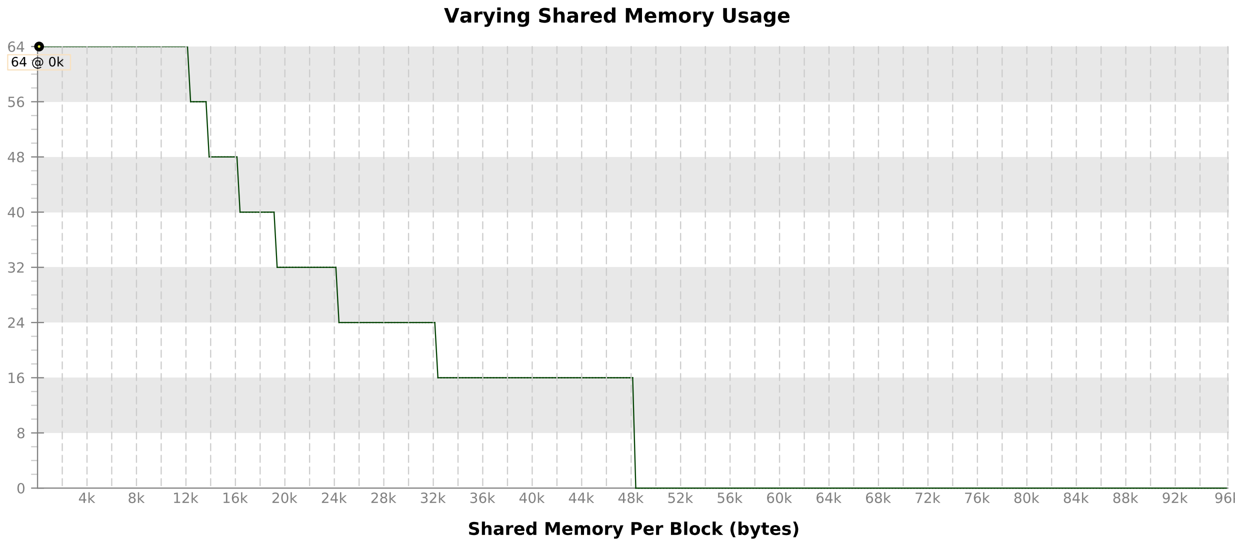
The following charts show how varying different components of the kernel will impact theoretical occupancy.



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

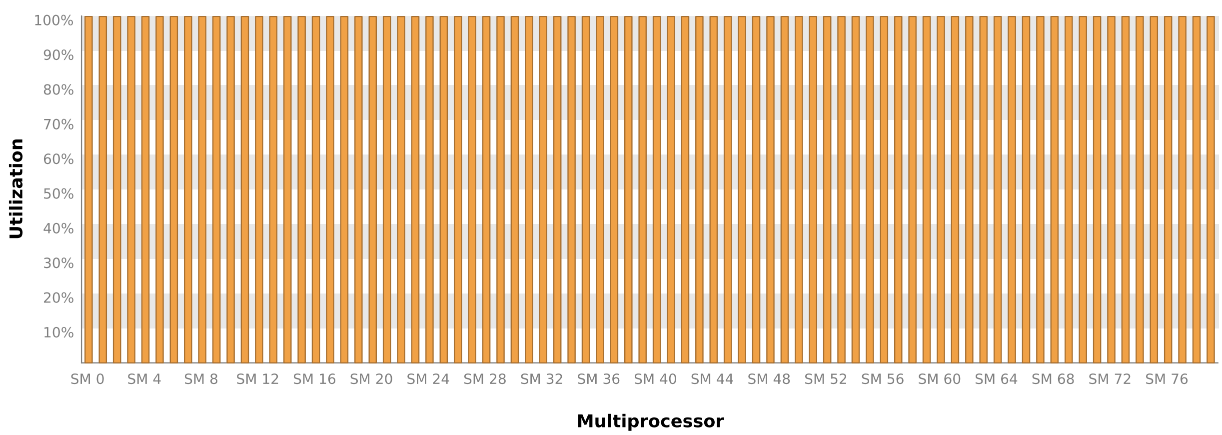


Varying Register Count
Registers Per Thread
64 @ 32
0
8
16
24
32
40
48
56
64
3744
7488
11232
14976
18720
22464
26208
29952
33696
37440
41184
44928
48672
52416
56160
59904
65536



2.3. 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.



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. 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 warp execution efficiency for these kernels is 84.8% if predicated instructions are not taken into account. The kernel's not predicated off warp execution efficiency of 76.6% is less than 100% due to divergent branches and predicated instructions.

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

3.2. 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.

/mxnet/src/operator/custom/./new-forward.cuh

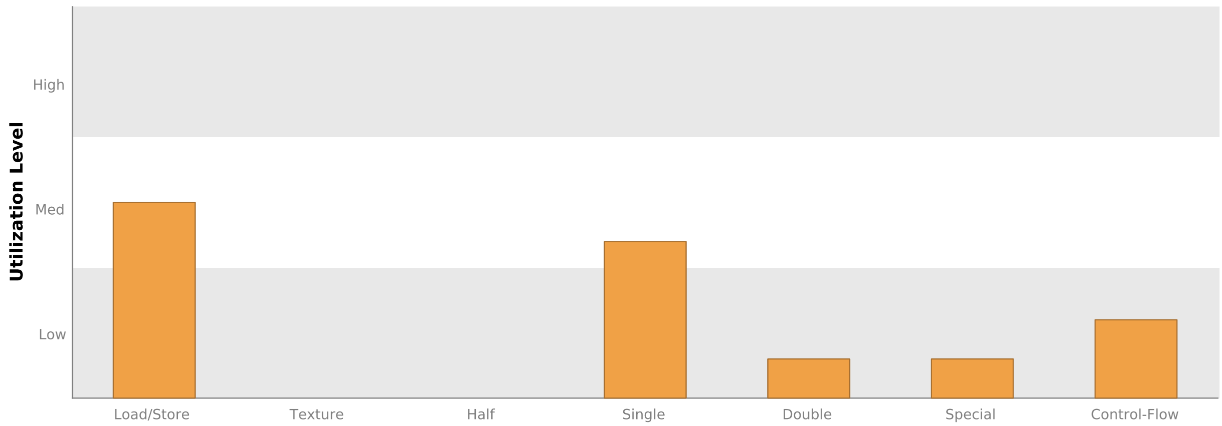
Line 39	Divergence = 16.5% [396000 divergent executions out of 2400000 total executions]
Line 42	Divergence = 0% [0 divergent executions out of 1980000 total executions]
Line 42	Divergence = 0% [0 divergent executions out of 1980000 total executions]
Line 43	Divergence = 0% [0 divergent executions out of 1980000 total executions]
Line 43	Divergence = 0% [0 divergent executions out of 9900000 total executions]
Line 44	Divergence = 0% [0 divergent executions out of 9900000 total executions]
Line 44	Divergence = 0% [0 divergent executions out of 9900000 total executions]
Line 44	Divergence = 0% [0 divergent executions out of 9900000 total executions]
Line 44	Divergence = 0% [0 divergent executions out of 9900000 total executions]
Line 44	Divergence = 0% [0 divergent executions out of 9900000 total executions]
Line 44	Divergence = 0% [0 divergent executions out of 9900000 total executions]
Line 45	Divergence = 0% [0 divergent executions out of 9900000 total executions]
Line 45	Divergence = 0% [0 divergent executions out of 9900000 total executions]
Line 54	Divergence = 0% [0 divergent executions out of 1980000 total executions]

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.

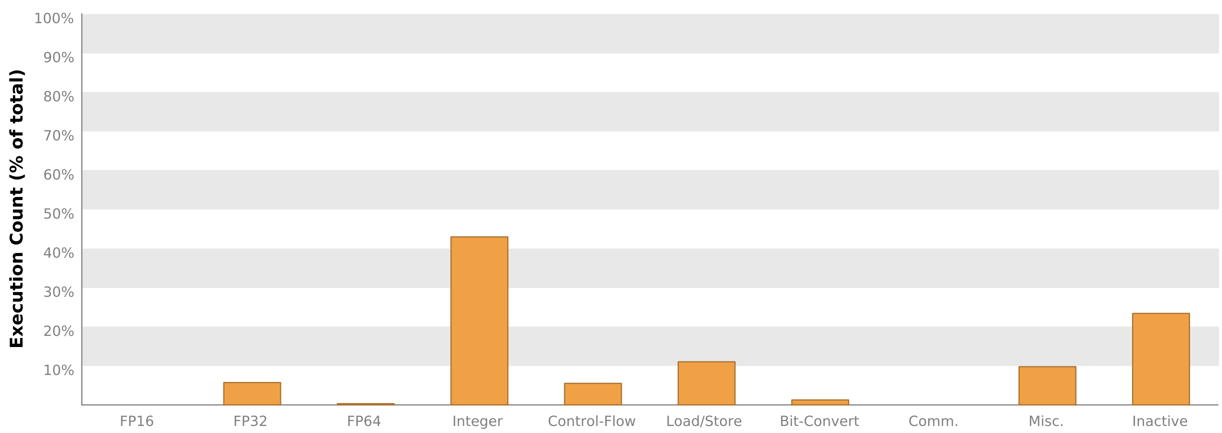
Load/Store - Load and store instructions for shared and constant memory.
Texture - Load and store instructions for local, global, and texture memory.
Half - Half-precision floating-point arithmetic instructions.
Single - Single-precision integer and floating-point arithmetic instructions.
Double - Double-precision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

Special - Special arithmetic instructions such as sin, cos, popc, etc.
Control-Flow - Direct and indirect branches, jumps, and calls.



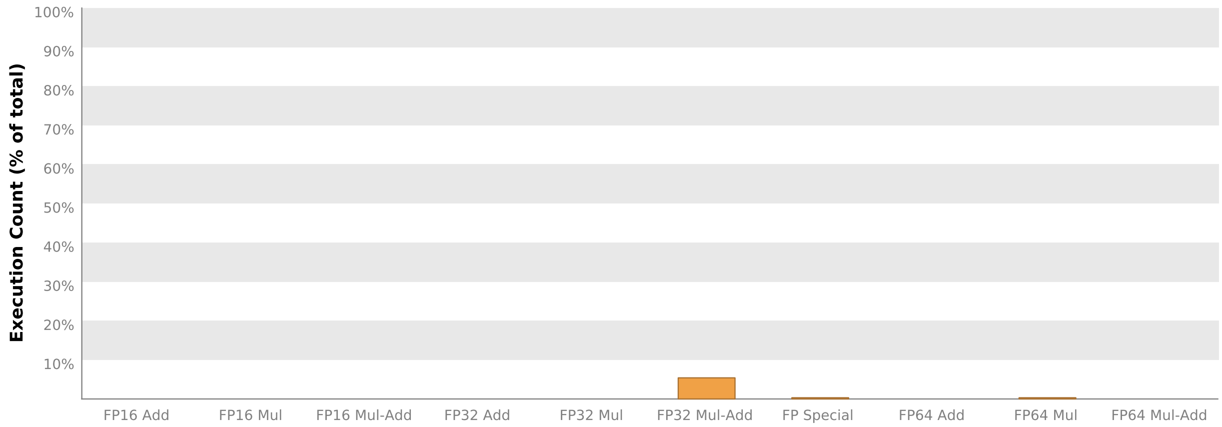
3.4. Instruction Execution Counts

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.



Execution Count (% of total)
100%
90%
80%
70%
60%
50%
40%
30%
20%
10%
FP16 Add
FP16 Mul
FP16 Mul-Add
FP32 Add
FP32 Mul
FP32 Mul-Add
FP Special
FP64 Add
FP64 Mul
FP64 Mul-Add

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	14924406	157.517 GB/s	
Writes	9504016	100.308 GB/s	
Total	24428422	257.825 GB/s	Idle Low Medium High Max
Unified Cache			
Local Loads	0	0 B/s	
Local Stores	0	0 B/s	
Global Loads	296990120	3,134.523 GB/s	
Global Stores	9504000	100.308 GB/s	
Texture Reads	128116378	5,408.716 GB/s	
Unified Total	434610498	8,643.547 GB/s	Idle Low Medium High Max
Device Memory			
Reads	17319730	182.798 GB/s	
Writes	9548378	100.776 GB/s	
Total	26868108	283.574 GB/s	Idle Low Medium High Max
System Memory			
[PCIe configuration: Gen3 x8, 8 Gbit/s]			
Reads	0	0 B/s	Idle Low Medium High Max
Writes	5	52.771 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.