Analysis Report

void IndexScalarGPU<unsigned int>(unsigned int const *, unsigned int, unsigned int*, unsigned int*)

Duration	28.08163 ms (28,081,633 ns)
Grid Size	[3907,1,1]
Block Size	[256,1,1]
Registers/Thread	32
Shared Memory/Block	32 B
Shared Memory Executed	2 KiB
Shared Memory Bank Size	4 B

[0] GeForce GTX 750 Ti

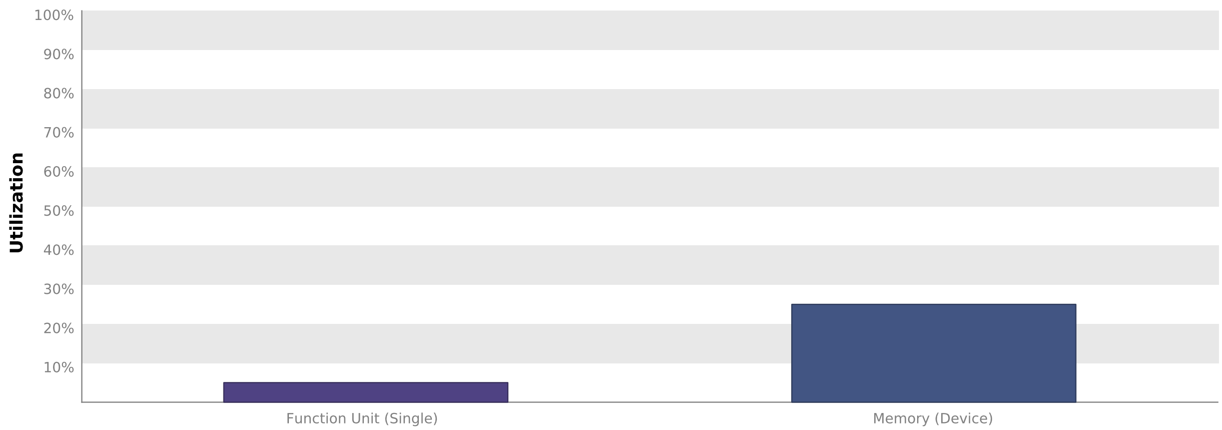
GPU UUID	GPU-12e9bc68-58bb-1f60-d768-e541cc24c6c5
Compute Capability	5.0
Max. Threads per Block	1024
Max. Threads per Multiprocessor	2048
Max. Shared Memory per Block	48 KiB
Max. Shared Memory per Multiprocessor	64 KiB
Max. Registers per Block	65536
Max. Registers per Multiprocessor	65536
Max. Grid Dimensions	[2147483647, 65535, 65535]
Max. Block Dimensions	[1024, 1024, 64]
Max. Warps per Multiprocessor	64
Max. Blocks per Multiprocessor	32
Single Precision FLOP/s	1.606 TeraFLOP/s
Double Precision FLOP/s	50.18 GigaFLOP/s
Number of Multiprocessors	5
Multiprocessor Clock Rate	1.254 GHz
Concurrent Kernel	true
Max IPC	6
Threads per Warp	32
Global Memory Bandwidth	86.4 GB/s
Global Memory Size	1.952 GiB
Constant Memory Size	64 KiB
L2 Cache Size	2 MiB
Memcpy Engines	1
PCIe Generation	2
PCIe Link Rate	5 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 "void IndexScalarGPU<unsigne..." 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 "GeForce GTX 750 Ti". 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.

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.

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.

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

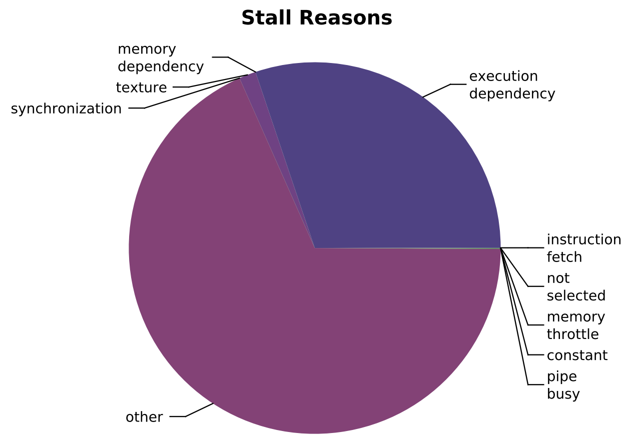
Memory Throttle - Large number of pending memory operations prevent further forward progress. These can be reduced by combining several memory transactions into one.

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

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.

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

Optimization: Resolve the primary stall issue; other.



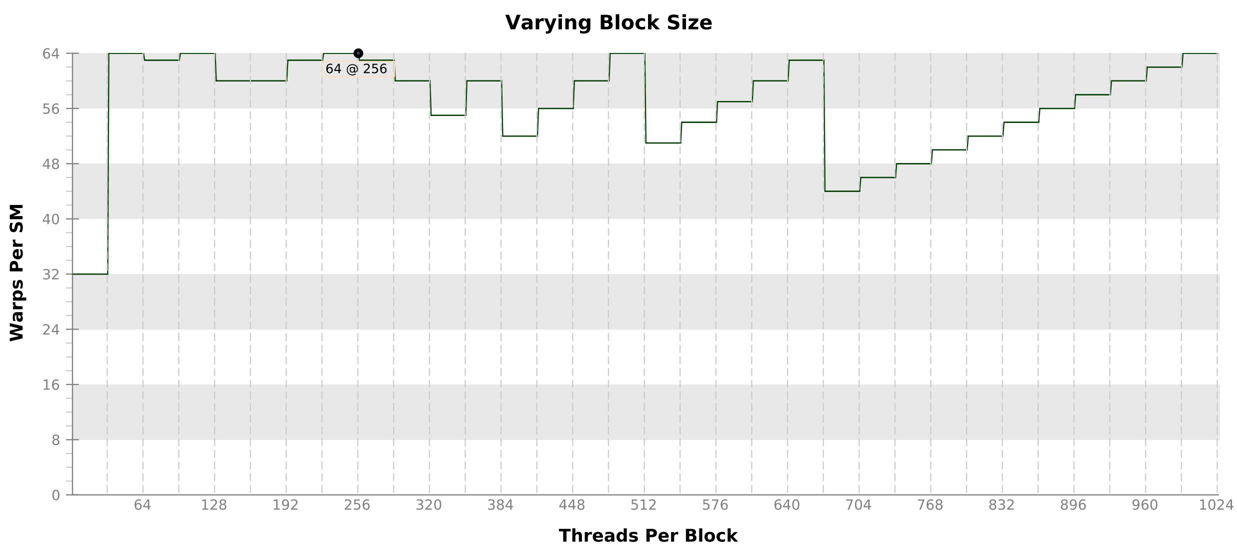
2.2. Occupancy Is Not Limiting Kernel Performance

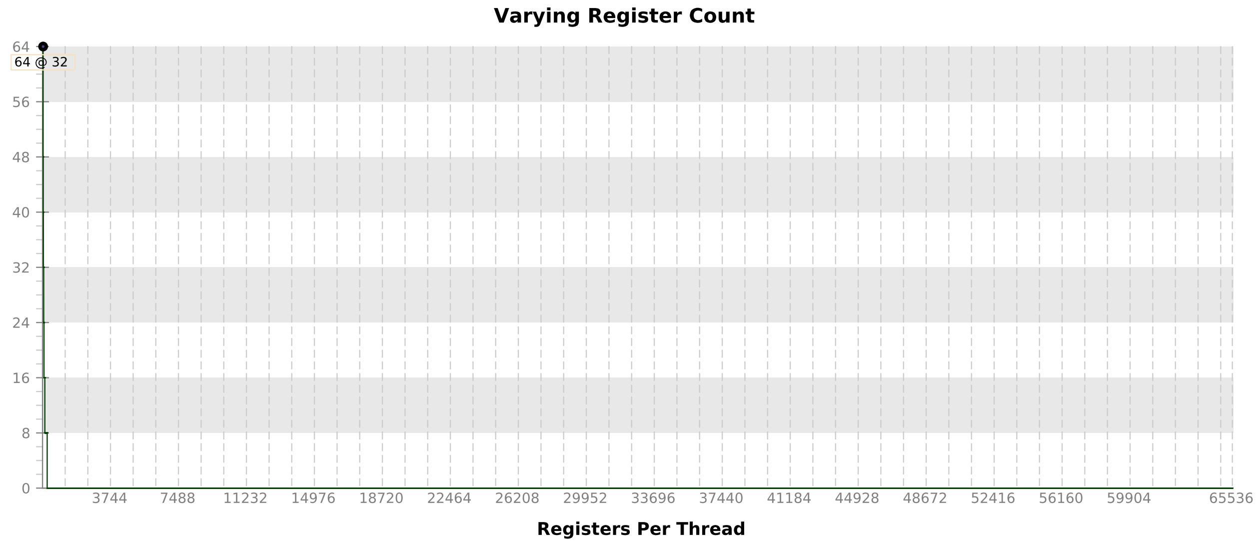
The kernel's block size, register usage, and shared memory usage allow it to fully utilize all warps on the GPU.

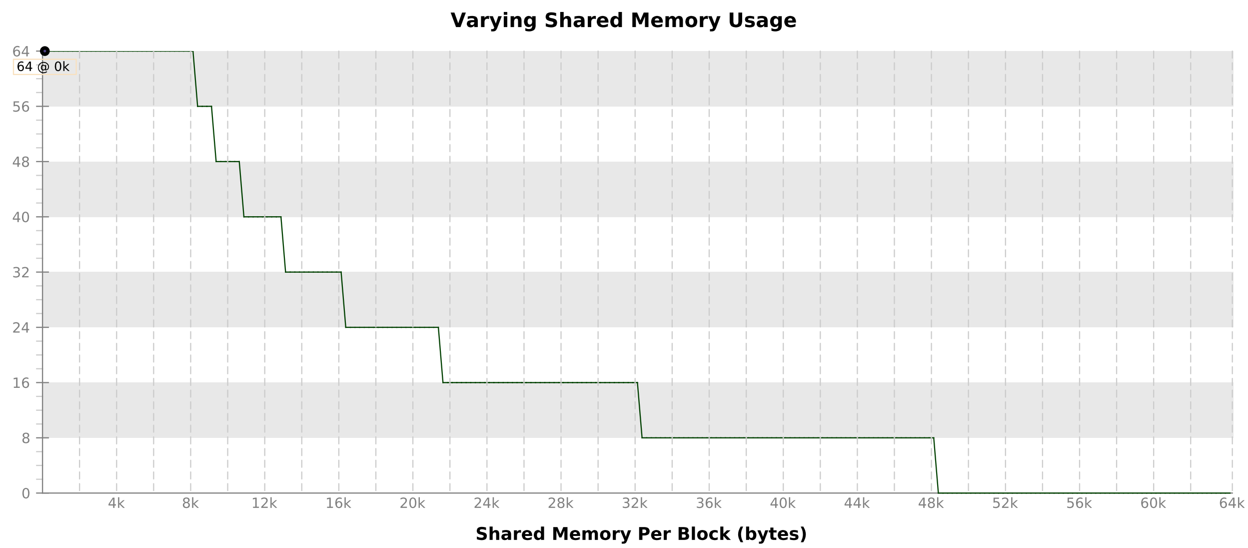
Variable	Achieved	Theoretical	Device Limit	Grid Size: [3907,1,1] (3907 blocks) Block Size: [256,1,
Occupancy Per SM				
Active Blocks		8	32	
Active Warps	54.44	64	64	
Active Threads		2048	2048	
Occupancy	85.1%	100%	100%	
Warps				
Threads/Block		256	1024	
Warps/Block		8	32	
Block Limit		8	32	
Registers				
Registers/Thread		32	65536	
Registers/Block		8192	65536	
Block Limit		8	32	
Shared Memory				
Shared Memory/Block		32	65536	
Block Limit		256	32	

2.3. Occupancy Charts

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

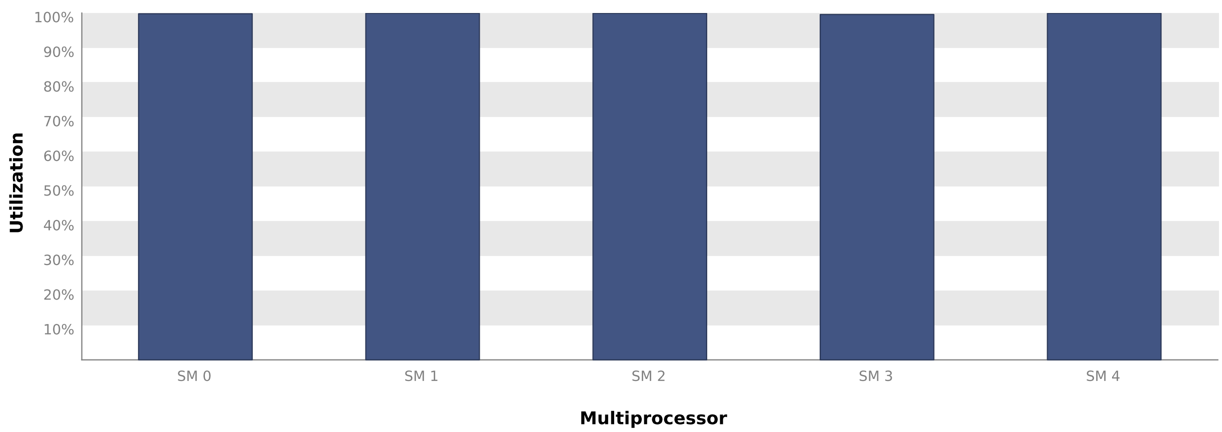






2.4. Multiprocessor Utilization

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.


100%
90%
80%
70%
60%
50%
40%
30%
20%
10%
Utilization
SM 0
SM 1
SM 2
SM 3
SM 4
Multiprocessor

3. Compute Resources

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

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 :

void IndexScalarGPU<unsigned int>(unsigned int const *, unsigned int, unsigned int*, unsigned int*)

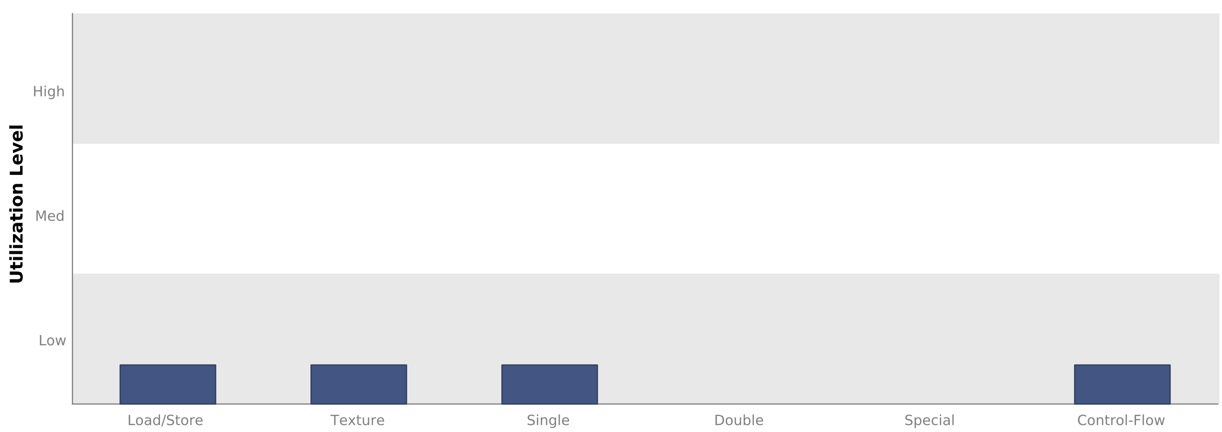
Maximum instruction execution count in assembly: 31256
Average instruction execution count in assembly: 30833
Instructions executed for the kernel: 13875054
Thread instructions executed for the kernel: 444001728
Non-predicated thread instructions executed for the kernel: 442001728
Warp non-predicated execution efficiency of the kernel: 99.5%
Warp execution efficiency of the kernel: 100.0%

Source files :

/home/francisco/Dropbox/StiffMa/Code/Scalar/IndexScalarsp.cu

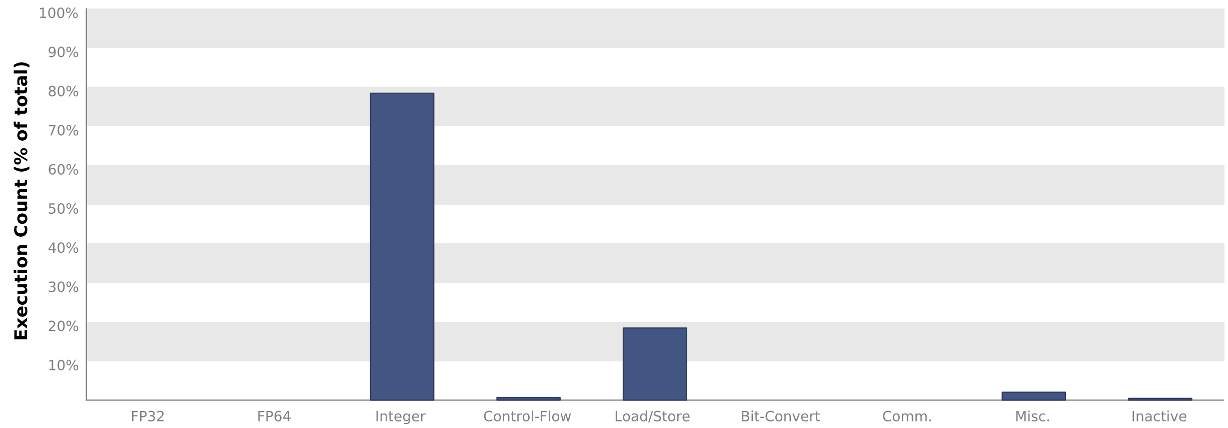
3.2. Function Unit Utilization

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.
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.



3.3. 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.



3.4. 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.

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	250000	1.264 GB/s	
Shared Total	250000	1.264 GB/s	Idle Low Medium High Max
L2 Cache			
Reads	8669381	10.958 GB/s	
Writes	72000006	91.006 GB/s	
Total	80669387	101.963 GB/s	Idle Low Medium High Max
Unified Cache			
Local Loads	0	0 B/s	
Local Stores	0	0 B/s	
Global Loads	8000000	10.112 GB/s	
Global Stores	72000000	91.006 GB/s	
Texture Reads	1000000	1.264 GB/s	
Unified Total	81000000	102.381 GB/s	Idle Low Medium High Max
Device Memory			
Reads	1715828	2.169 GB/s	
Writes	13031811	16.472 GB/s	
Total	14747639	18.641 GB/s	Idle Low Medium High Max
System Memory			
[PCIe configuration: Gen2 x16, 5 Gbit/s]			
Reads	0	0 B/s	Idle Low Medium High Max
Writes	5	6.319 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.