

# Analysis Report

**conwayThread(char\*, char\*, int, int)**

Duration	211.322 $\mu$ s
Grid Size	[ 64,1,1 ]
Block Size	[ 1024,1,1 ]
Registers/Thread	22
Shared Memory/Block	0 B
Shared Memory Requested	48 KiB
Shared Memory Executed	48 KiB
Shared Memory Bank Size	4 B

## [0] GeForce GTX 650 Ti

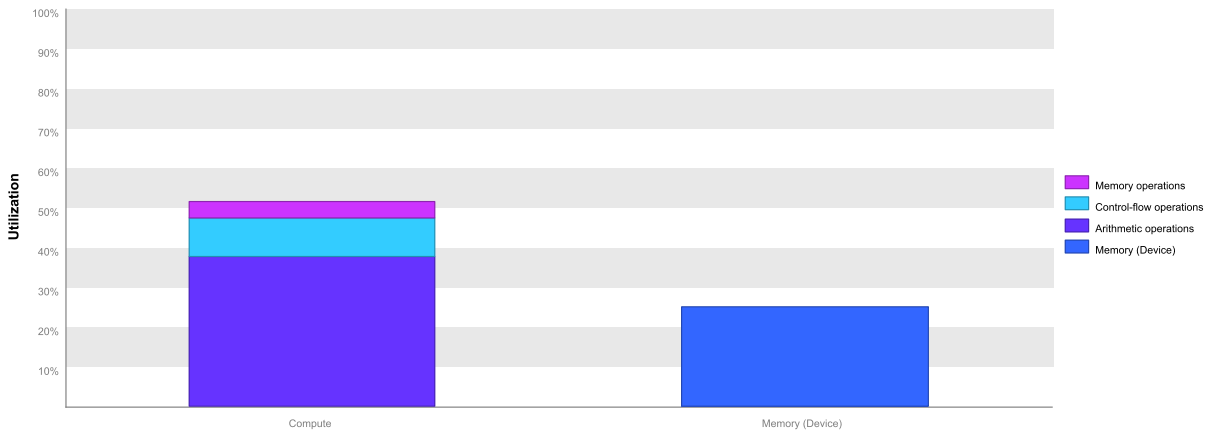
Compute Capability	3.0
Max. Threads per Block	1024
Max. Shared Memory per Block	48 KiB
Max. Registers per Block	65536
Max. Grid Dimensions	[ 2147483647, 65535, 65535 ]
Max. Block Dimensions	[ 1024, 1024, 64 ]
Max. Warps per Multiprocessor	64
Max. Blocks per Multiprocessor	16
Number of Multiprocessors	4
Multiprocessor Clock Rate	980 MHz
Concurrent Kernel	true
Max IPC	7
Threads per Warp	32
Global Memory Bandwidth	86.4 GB/s
Global Memory Size	1 GiB
Constant Memory Size	64 KiB
L2 Cache Size	256 KiB
Memcpy Engines	1
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 "conwayThread" 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 650 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.

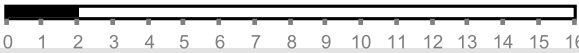


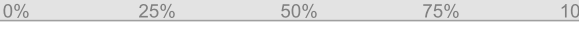

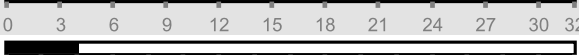
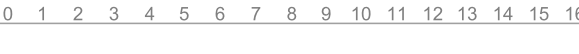


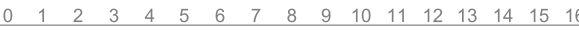



## 2. Instruction and Memory Latency

Instruction and memory latency limit the performance of a kernel when the GPU does not have enough work to keep busy. The 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.

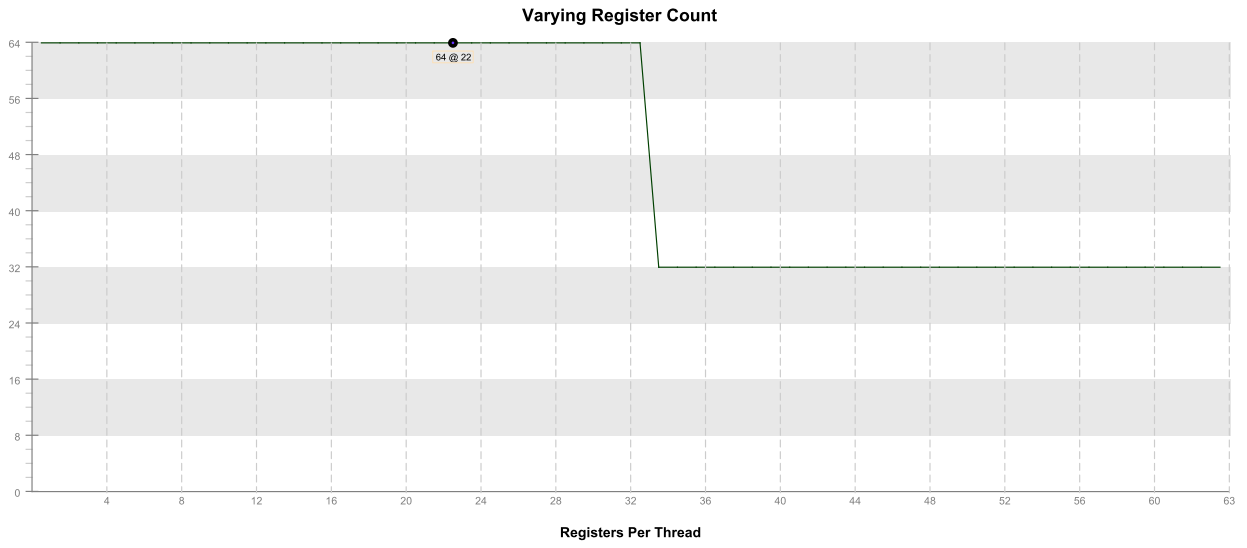
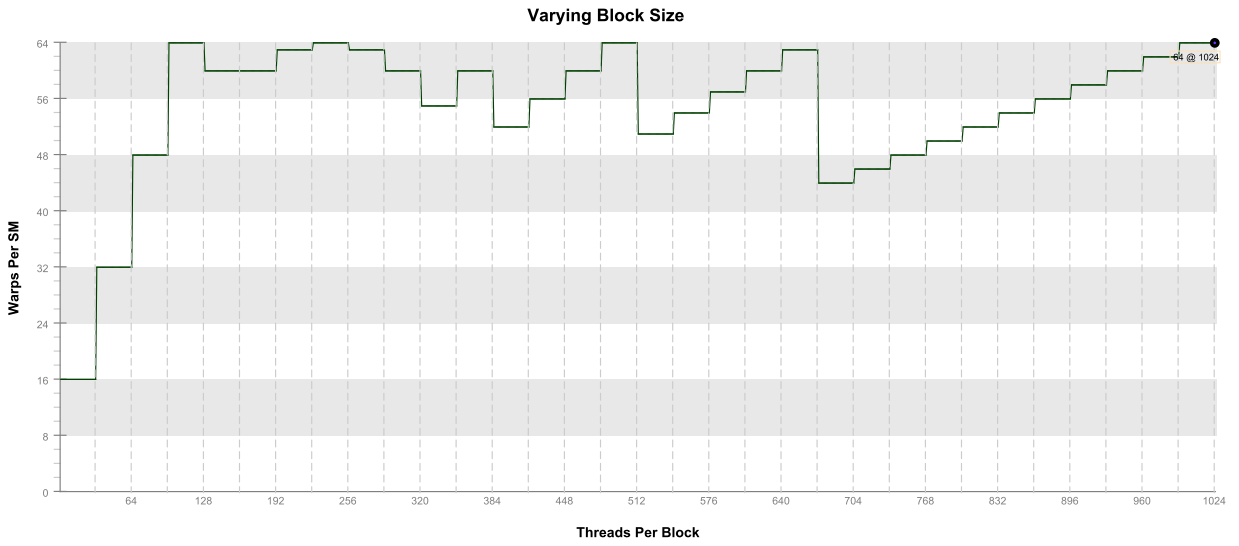
### 2.1. Occupancy Is Not Limiting Kernel Performance

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

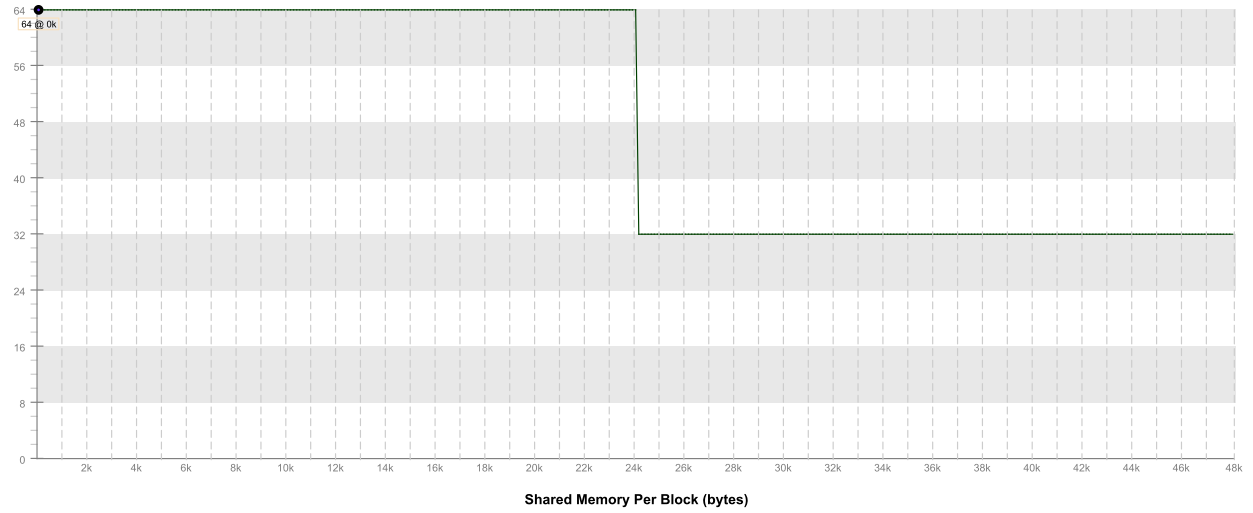
Variable	Achieved	Theoretical	Device Limit	Grid Size: [ 64,1,1 ] (64 blocks) Block Size: [ 1024,1,1 ] (1024 threads)
Occupancy Per SM				
Active Blocks		2	16	
Active Warps	61.71	64	64	
Active Threads		2048	2048	
Occupancy	96.4%	100%	100%	
Warps				
Threads/Block		1024	1024	
Warps/Block		32	32	
Block Limit		2	16	
Registers				
Registers/Thread		22	63	
Registers/Block		24576	65536	
Block Limit		2	16	
Shared Memory				
Shared Memory/Block		0	49152	
Block Limit			16	

### 2.2. Occupancy Charts

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



Varying Shared Memory Usage



### 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

The kernel profile 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.

*Optimization: Select a kernel or source file listed below to view the profile. Examine portions of the kernel that have high execution counts and inactive or predicated threads to identify optimization opportunities.*

#### 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.*

file:/F:/Documents/CS%20530/CS530hw/Lab2VS/cuda\_project/cuda\_project/kernel.cu

Line 56	Divergence = 12.5% [ 256 divergent executions out of 2048 total executions ]
Line 61	Divergence = 84.2% [ 1725 divergent executions out of 2048 total executions ]
Line 69	Divergence = 12.5% [ 256 divergent executions out of 2048 total executions ]
Line 74	Divergence = 84.2% [ 1725 divergent executions out of 2048 total executions ]
Line 85	Divergence = 84.2% [ 1725 divergent executions out of 2048 total executions ]
Line 92	Divergence = 12.5% [ 256 divergent executions out of 2048 total executions ]
Line 97	Divergence = 84.2% [ 1725 divergent executions out of 2048 total executions ]
Line 102	Divergence = 12.5% [ 256 divergent executions out of 2048 total executions ]
Line 107	Divergence = 84.2% [ 1725 divergent executions out of 2048 total executions ]
Line 114	Divergence = 12.5% [ 256 divergent executions out of 2048 total executions ]
Line 119	Divergence = 84.2% [ 1725 divergent executions out of 2048 total executions ]
Line 129	Divergence = 84.2% [ 1725 divergent executions out of 2048 total executions ]
Line 136	Divergence = 12.5% [ 256 divergent executions out of 2048 total executions ]
Line 141	Divergence = 84.2% [ 1725 divergent executions out of 2048 total executions ]
Line 144	Divergence = 84.2% [ 1725 divergent executions out of 2048 total executions ]
Line 146	Divergence = 59.5% [ 1026 divergent executions out of 1725 total executions ]
Line 146	Divergence = 34.6% [ 597 divergent executions out of 1725 total executions ]
Line 157	Divergence = 62.9% [ 1288 divergent executions out of 2048 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

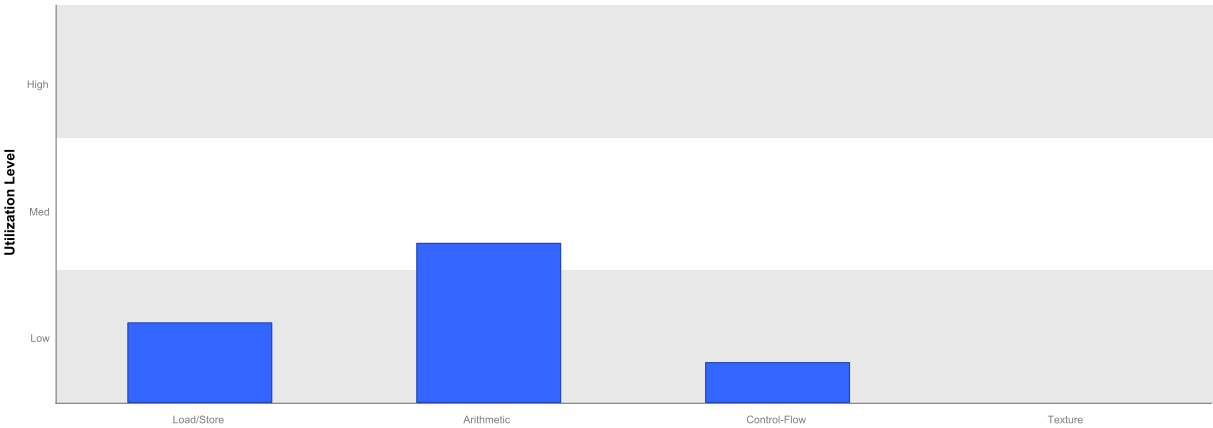
by overuse of any function unit.

Load/Store - Load and store instructions for local, shared, global, constant, etc. memory.

Arithmetic - All arithmetic instructions including integer and floating-point add and multiply, logical and binary operations, etc.

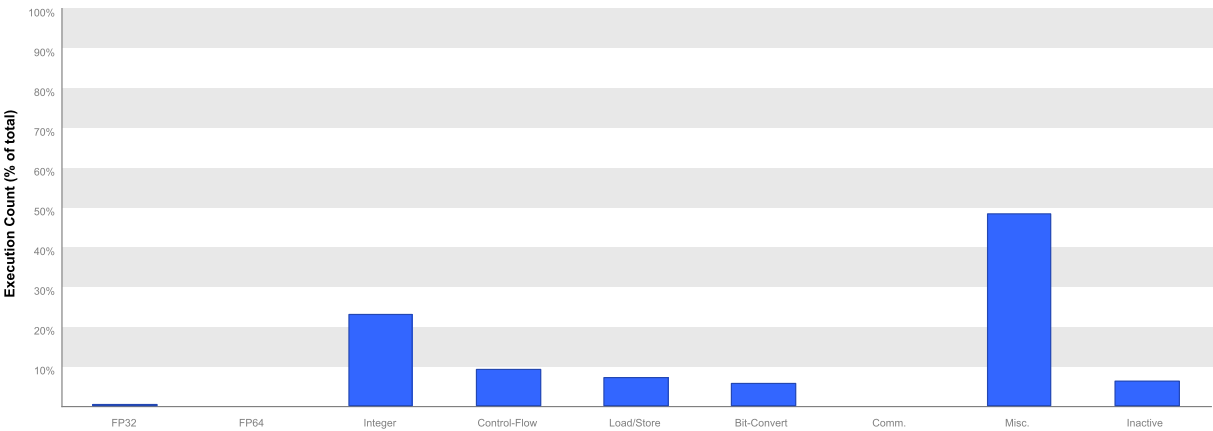
Control-Flow - Direct and indirect branches, jumps, and calls.

Texture - Texture operations.



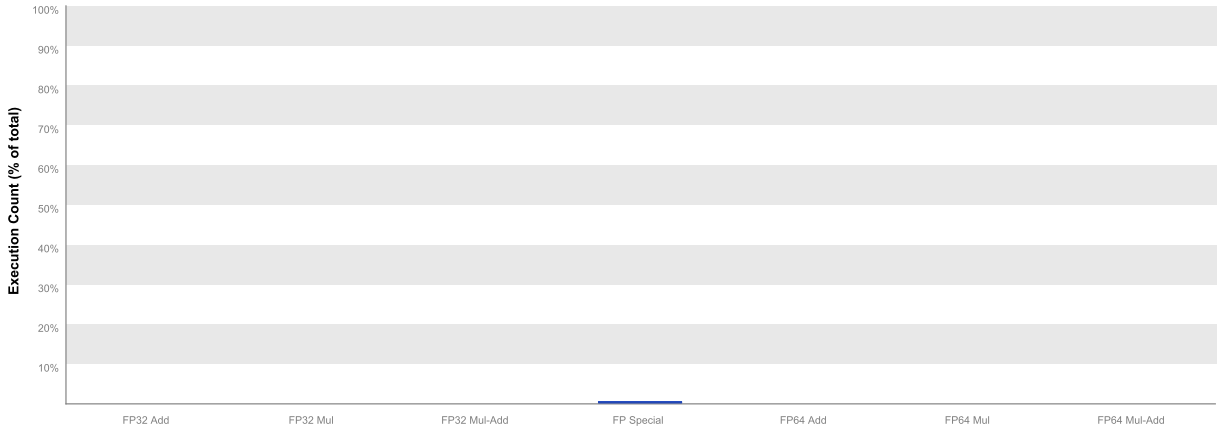
### 3.4. Instruction Execution Counts

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
L1/Shared Memory			
Local Loads	53371	32.069 GB/s	
Local Stores	51896	13.225 GB/s	
Shared Loads	0	0 B/s	
Shared Stores	0	0 B/s	
Global Loads	21504	4.625 GB/s	
Global Stores	6087	916.471 MB/s	
Atomic	0	0 B/s	
L1/Shared Total	132858	50.835 GB/s	
L2 Cache			
L1 Reads	166340	25.044 GB/s	
L1 Writes	124527	18.749 GB/s	
Texture Reads	0	0 B/s	
Atomic	0	0 B/s	
Total	290867	43.794 GB/s	
Texture Cache			
Reads	0	0 B/s	
Device Memory			
Reads	6513	980.61 MB/s	
Writes	123087	18.532 GB/s	
Total	129600	19.513 GB/s	
System Memory			
[ PCIe configuration: Gen3 x16, 8 Gbit/s ]			
Reads	0	0 B/s	
Writes	5	752.81 kB/s	