

Analysis Report

cuda_kernel(int*, int*, unsigned long, unsigned long)

| | |
|-------------------------|---------------------------|
| Duration | 54.105 ms (54,104,903 ns) |
| Grid Size | [626,626,1] |
| Block Size | [16,16,1] |
| Registers/Thread | 22 |
| Shared Memory/Block | 0 B |
| Shared Memory Requested | 96 KiB |
| Shared Memory Executed | 96 KiB |
| Shared Memory Bank Size | 4 B |

[0] GeForce GTX 970

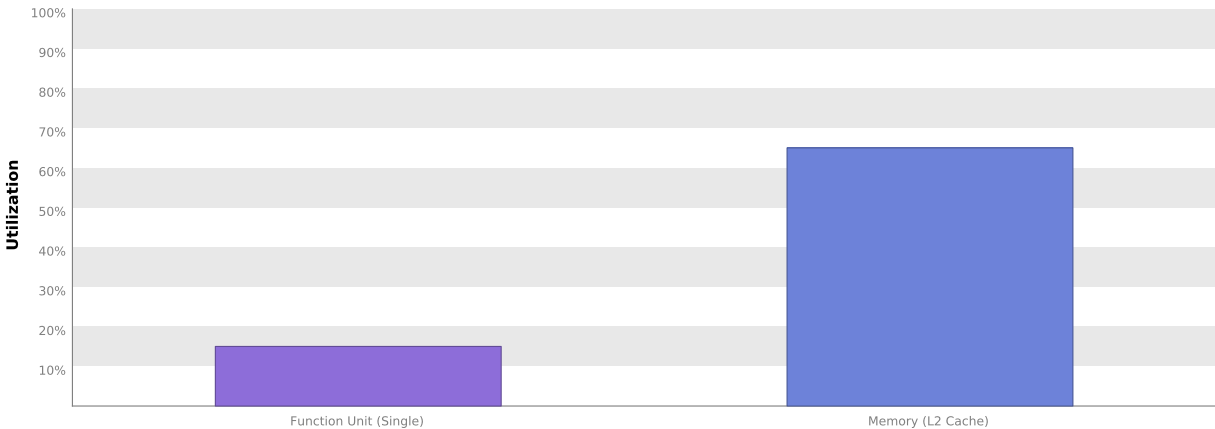
| | |
|---------------------------------------|--|
| GPU UUID | GPU-92461dda-c508-5f62-ee5f-85fc2de903e6 |
| Compute Capability | 5.2 |
| Max. Threads per Block | 1024 |
| Max. Threads per Multiprocessor | 2048 |
| Max. Shared Memory per Block | 48 KiB |
| Max. Shared Memory per Multiprocessor | 96 KiB |
| 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 |
| Single Precision FLOP/s | 4.17 TeraFLOP/s |
| Double Precision FLOP/s | 130.312 GigaFLOP/s |
| Number of Multiprocessors | 13 |
| Multiprocessor Clock Rate | 1.253 GHz |
| Concurrent Kernel | true |
| Max IPC | 6 |
| Threads per Warp | 32 |
| Global Memory Bandwidth | 224.32 GB/s |
| Global Memory Size | 3.942 GiB |
| Constant Memory Size | 64 KiB |
| L2 Cache Size | 1.75 MiB |
| Memcpy Engines | 2 |
| 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 "cuda_kernel" is most likely limited by memory bandwidth. You should first examine the information in the "Memory Bandwidth" section to determine how it is limiting performance.

1.1. Kernel Performance Is Bound By Memory Bandwidth

For device "GeForce GTX 970" the kernel's compute utilization is significantly lower than its memory utilization. These utilization levels indicate that the performance of the kernel is most likely being limited by the memory system. For this kernel the limiting factor in the memory system is the bandwidth of the L2 Cache memory.



2. 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. The results below indicate that the kernel is limited by the bandwidth available to the L2 cache.

2.1. Global Memory Alignment and Access Pattern

Memory bandwidth is used most efficiently when each global memory load and store has proper alignment and access pattern.

Optimization: Each entry below points to a global load or store within the kernel with an inefficient alignment or access pattern. For each load or store improve the alignment and access pattern of the memory access.

2.2. GPU Utilization Is Limited By Memory Bandwidth

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. The results show that the kernel's performance is potentially limited by the bandwidth available from one or more of the memories on the device.

Optimization: Try the following optimizations for the memory with high bandwidth utilization.

Shared Memory - If possible use 64-bit accesses to shared memory and 8-byte bank mode to achieved 2x throughput.

L2 Cache - Align and block kernel data to maximize L2 cache efficiency.

Unified Cache - Reallocate texture data to shared or global memory. Resolve alignment and access pattern issues for global loads and stores.

Device Memory - Resolve alignment and access pattern issues for global loads and stores.

System Memory (via PCIe) - Make sure performance critical data is placed in device or shared memory.

| | | | |
|--|------------|--------------|--|
| Transactions | Bandwidth | Utilization | |
| Shared Memory | | | |
| Shared Loads | 0 | 0 B/s | |
| Shared Stores | 0 | 0 B/s | |
| Shared Total | 0 | 0 B/s | |
| L2 Cache | | | |
| Reads | 479414668 | 334.718 GB/s | |
| Writes | 98445008 | 68.732 GB/s | |
| Total | 577859676 | 403.45 GB/s | |
| Unified Cache | | | |
| Local Loads | 0 | 0 B/s | |
| Local Stores | 0 | 0 B/s | |
| Global Loads | 803809367 | 334.17 GB/s | |
| Global Stores | 98444952 | 68.732 GB/s | |
| Texture Reads | 103156807 | 72.022 GB/s | |
| Unified Total | 1005411126 | 474.924 GB/s | |
| Device Memory | | | |
| Reads | 25610027 | 17.88 GB/s | |
| Writes | 12514854 | 8.738 GB/s | |
| Total | 38124881 | 26.618 GB/s | |
| System Memory | | | |
| [PCIe configuration: Gen2 x16, 5 Gbit/s] | | | |
| Reads | 0 | 0 B/s | |
| Writes | 5 | 3.49 kB/s | |

3. 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.

3.1. Kernel Profile - PC Sampling

The Kernel Profile - PC Sampling gives the number of samples for each source and assembly line with various stall reasons. 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.

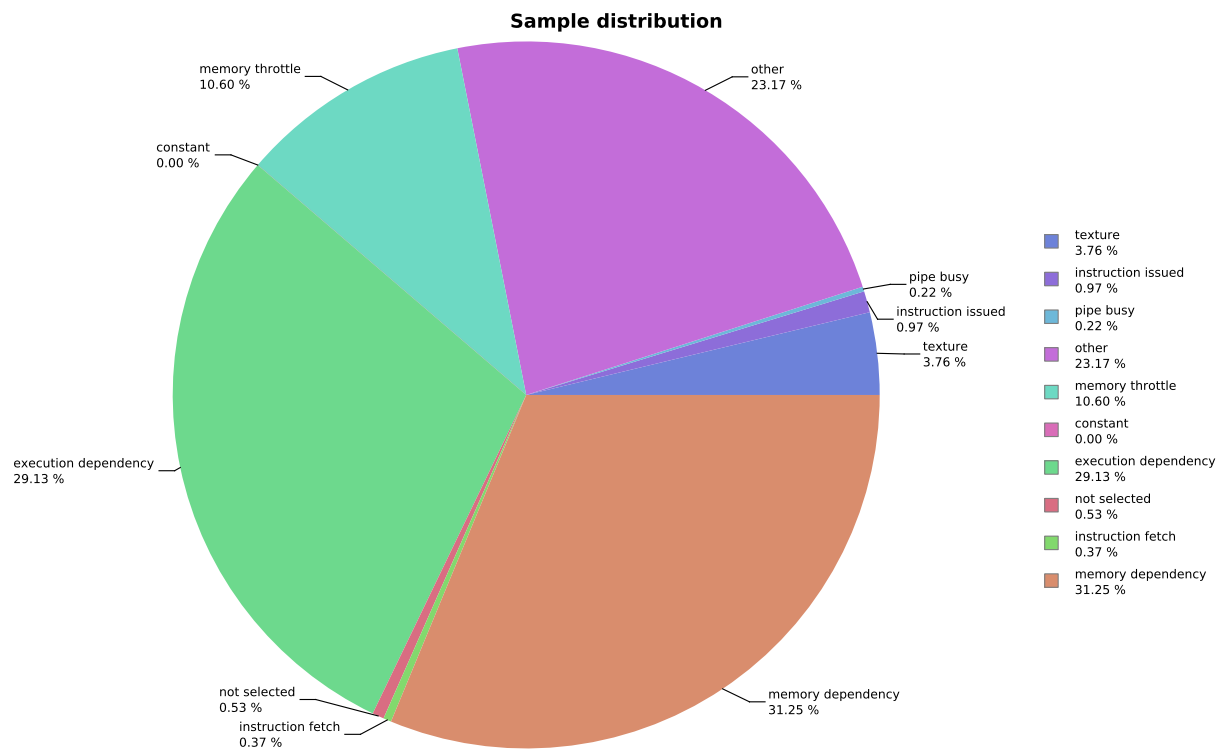
Memory Throttle - Large number of pending memory operations prevent further forward progress. These can be reduced by combining several memory transactions into one.

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 a uncommon reason.

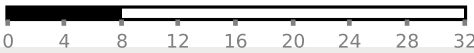


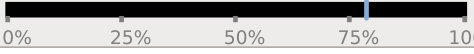

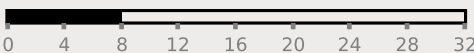
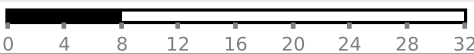
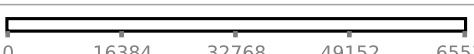


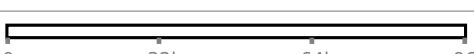
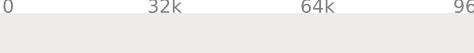
Examine portions of the kernel that have high number of samples to know where the maximum time was spent and observe the latency reasons for those samples to identify optimization opportunities.

| Cuda Functions | Sample Count | % of Kernel Samples |
|---|--------------|---------------------|
| cuda_kernel(int*, int*, unsigned long, unsigned long) | 3571327 | 100.0 |



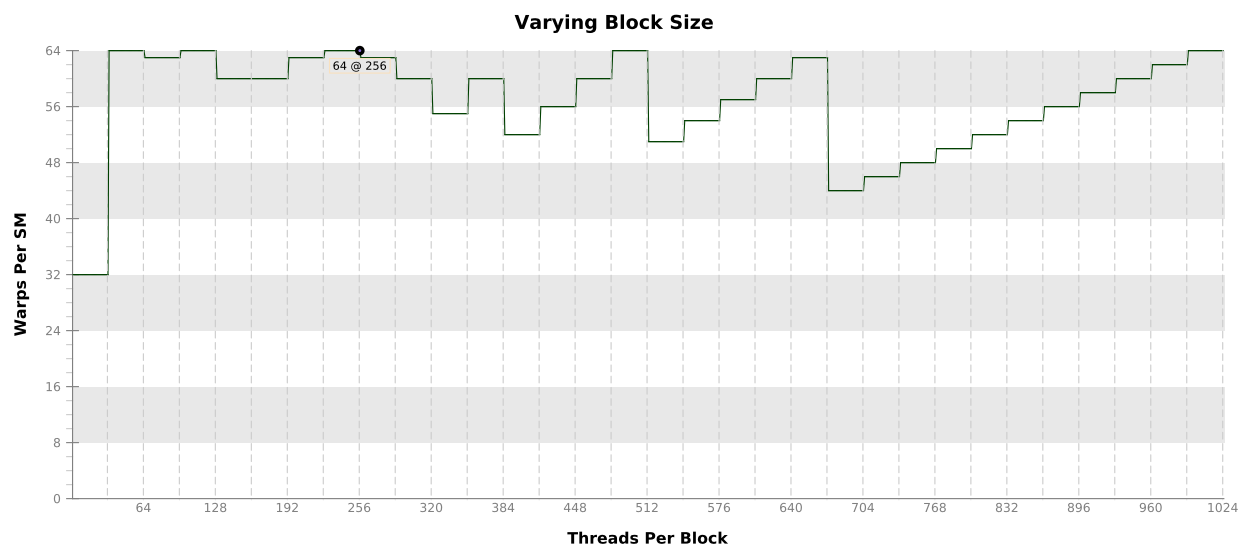
3.2. Occupancy Is Not Limiting Kernel Performance

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

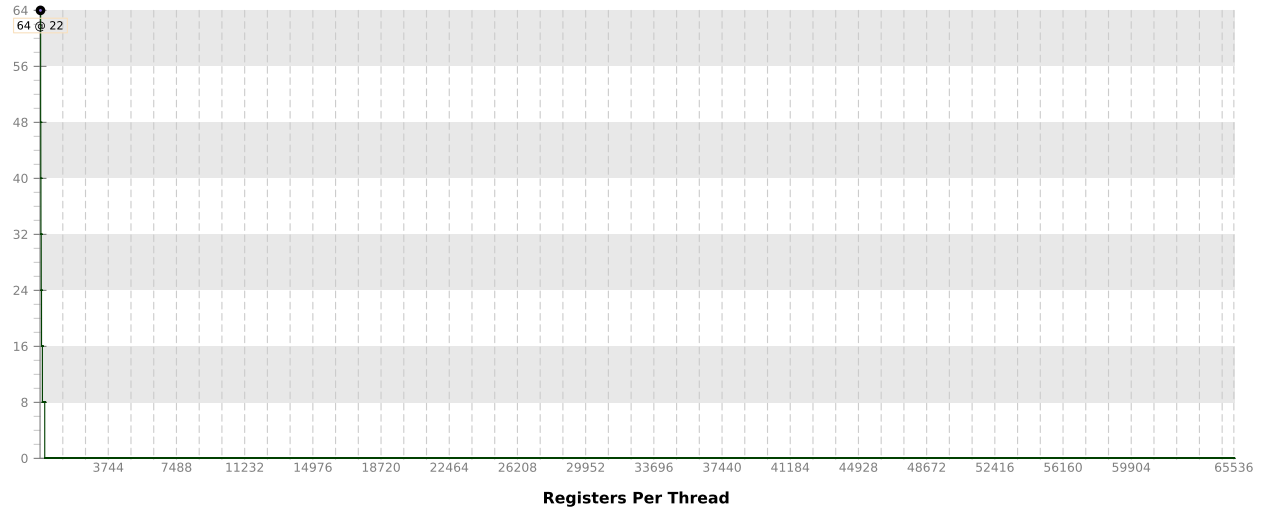
| Variable | Achieved | Theoretical | Device Limit | Grid Size: [626,626,1] (391876 blocks) Block Size: [16 |
|---------------------|----------|-------------|--------------|--|
| Occupancy Per SM | | | | |
| Active Blocks | | 8 | 32 |  |
| Active Warps | 49.96 | 64 | 64 |  |
| Active Threads | | 2048 | 2048 |  |
| Occupancy | 78.1% | 100% | 100% |  |
| Warps | | | | |
| Threads/Block | | 256 | 1024 |  |
| Warps/Block | | 8 | 32 |  |
| Block Limit | | 8 | 32 |  |
| Registers | | | | |
| Registers/Thread | | 22 | 65536 |  |
| Registers/Block | | 6144 | 65536 |  |
| Block Limit | | 10 | 32 |  |
| Shared Memory | | | | |
| Shared Memory/Block | | 0 | 98304 |  |
| Block Limit | | | 32 |  |

3.3. Occupancy Charts

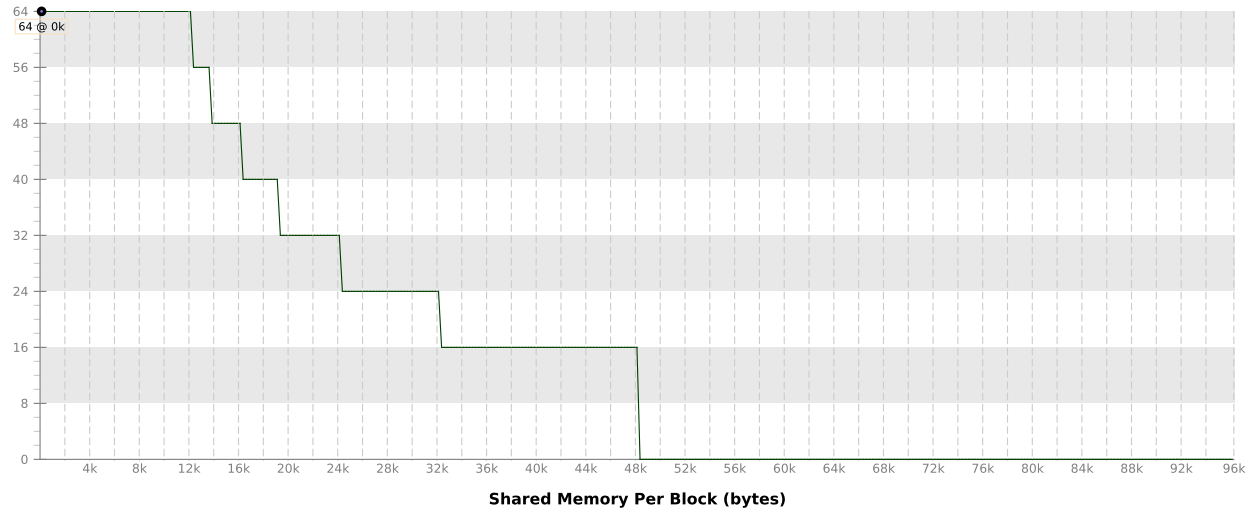
The following charts show how varying different components of the kernel will impact theoretical occupancy.



Varying Register Count



Varying Shared Memory Usage



4. 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.

4.1. 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 Functions :

| |
|---|
| cuda_kernel(int*, int*, unsigned long, unsigned long) |
|---|

Maximum instruction execution count in assembly: 3135008

Average instruction execution count in assembly: 2945843

Instructions executed for the kernel: 388851401

Thread instructions executed for the kernel: 11945523722

Non-predicated thread instructions executed for the kernel: 11724967975

Warp non-predicated execution efficiency of the kernel: 94.2%

Warp execution efficiency of the kernel: 96.0%

4.2. 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.

4.3. 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.

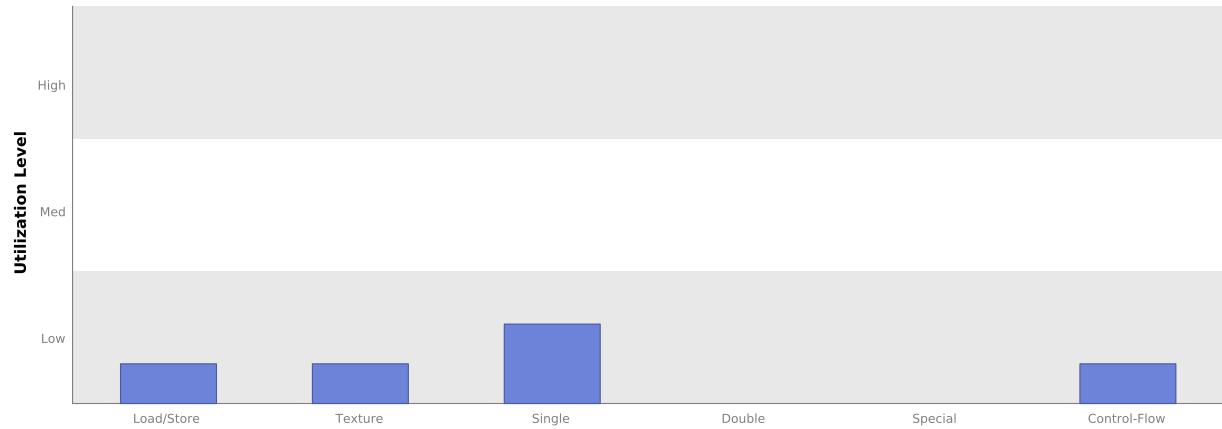
Texture - Load and store instructions for local, global, and texture memory.

Single - Single-precision integer and floating-point arithmetic instructions.

Double - Double-precision floating-point arithmetic instructions.

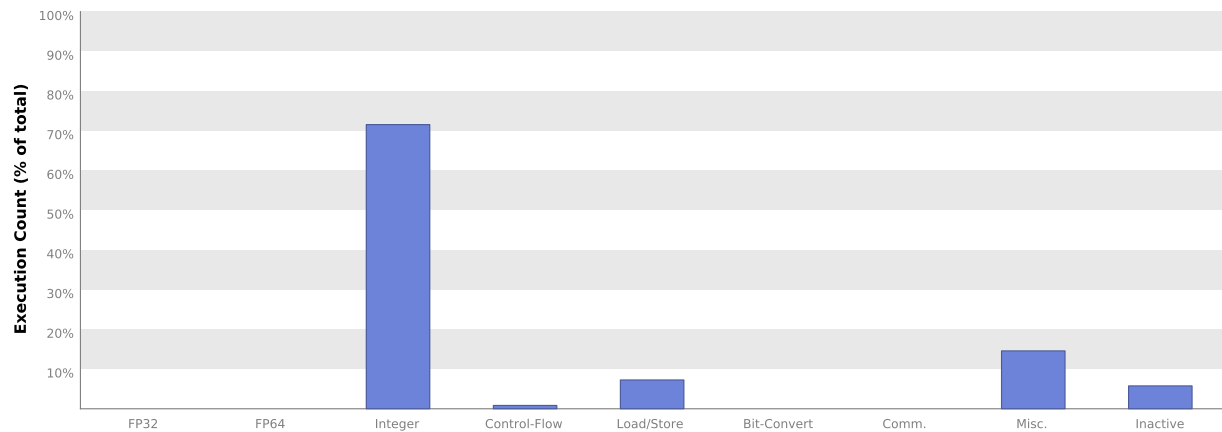
Special - Special arithmetic instructions such as sin, cos, popc, etc.

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



4.4. 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.



4.5. 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.

