

Analysis Report

void computeHOGlocal<float, float, float, int=8, int=8, int=16, int=16, int=64>(float*, float*, float*, float*, int, int, int, int)

| | |
|-------------------------|-------------------------|
| Duration | 3.276 ms (3,275,868 ns) |
| Grid Size | [31,1,1] |
| Block Size | [256,1,1] |
| Registers/Thread | 72 |
| Shared Memory/Block | 0 B |
| Shared Memory Requested | 96 KiB |
| Shared Memory Executed | 96 KiB |
| Shared Memory Bank Size | 4 B |

[0] GeForce GTX 960

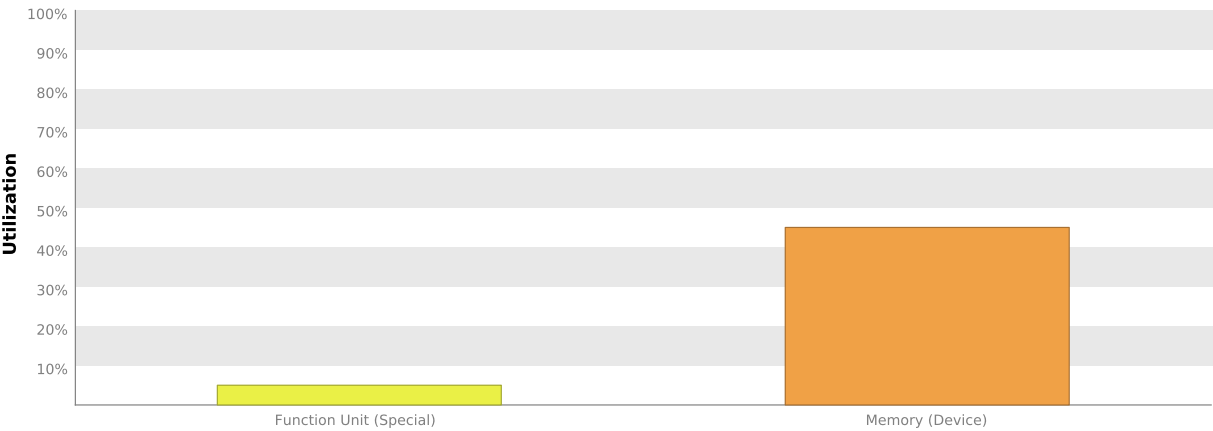
| | |
|--------------------------------|--|
| GPU UUID | GPU-0db32734-f94e-48a7-8b5d-4604317dc554 |
| Compute Capability | 5.2 |
| Max. Threads per Block | 1024 |
| Max. Shared Memory per Block | 48 KiB |
| Max. Registers per Block | 65536 |
| Max. Grid Dimensions | [2147483647, 65535, 65535] |
| Max. Block Dimensions | [1024, 1024, 64] |
| Max. Warps per Multiprocessor | 64 |
| Max. Blocks per Multiprocessor | 32 |
| Single Precision FLOP/s | 2.644 TeraFLOP/s |
| Double Precision FLOP/s | 82.624 GigaFLOP/s |
| Number of Multiprocessors | 8 |
| Multiprocessor Clock Rate | 1.291 GHz |
| Concurrent Kernel | true |
| Max IPC | 6 |
| Threads per Warp | 32 |
| Global Memory Bandwidth | 112.16 GB/s |
| Global Memory Size | 4 GiB |
| Constant Memory Size | 64 KiB |
| L2 Cache Size | 1 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 "void computeHOGlocal<float,..." 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 960". 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. 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.

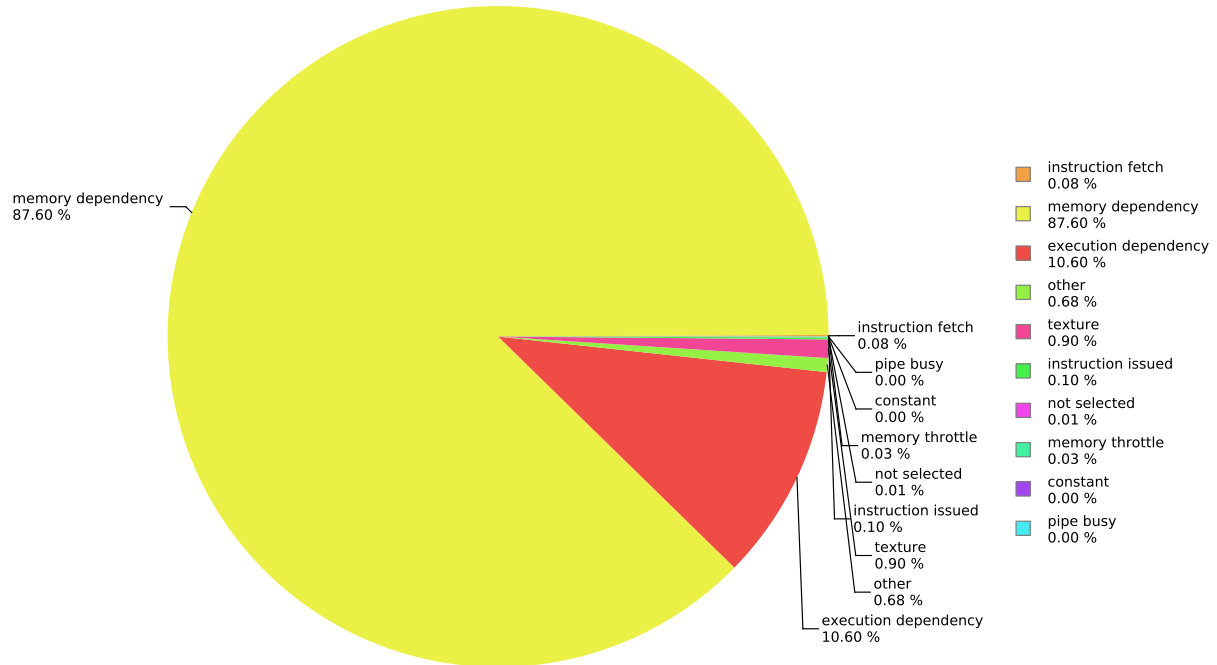
Examine portions of the kernel that have high number of samples to know where the maximum time was spent and observe the latency reasons for those samples to identify optimization opportunities.

| Cuda Functions | Sample Count | % of Kernel Samples |
|--|--------------|---------------------|
| <code>void computeHOGlocal<float, float, float, int=8, int=8, int=16, int=16, int=64>(float*, float*, float*, float*, int, int, int, int)</code> | 197468 | 2.119163508422491 |

Source Files :

| |
|--|
| <code>/home/adas/cuda-workspace/CudaVisionSysDeploy/Release/./src/init/./device/HOG/normalizeDescriptor.cuh</code> |
| <code>/home/adas/cuda-workspace/CudaVisionSysDeploy/Release/./src/init/./device/HOG/HOGdescriptor.cuh</code> |
| <code>/home/adas/cuda-workspace/CudaVisionSysDeploy/Release/./src/init/./device/HOG/addToHistogram.cuh</code> |
| <code>/usr/local/cuda-7.5/targets/x86_64-linux/include/device_functions.hpp</code> |

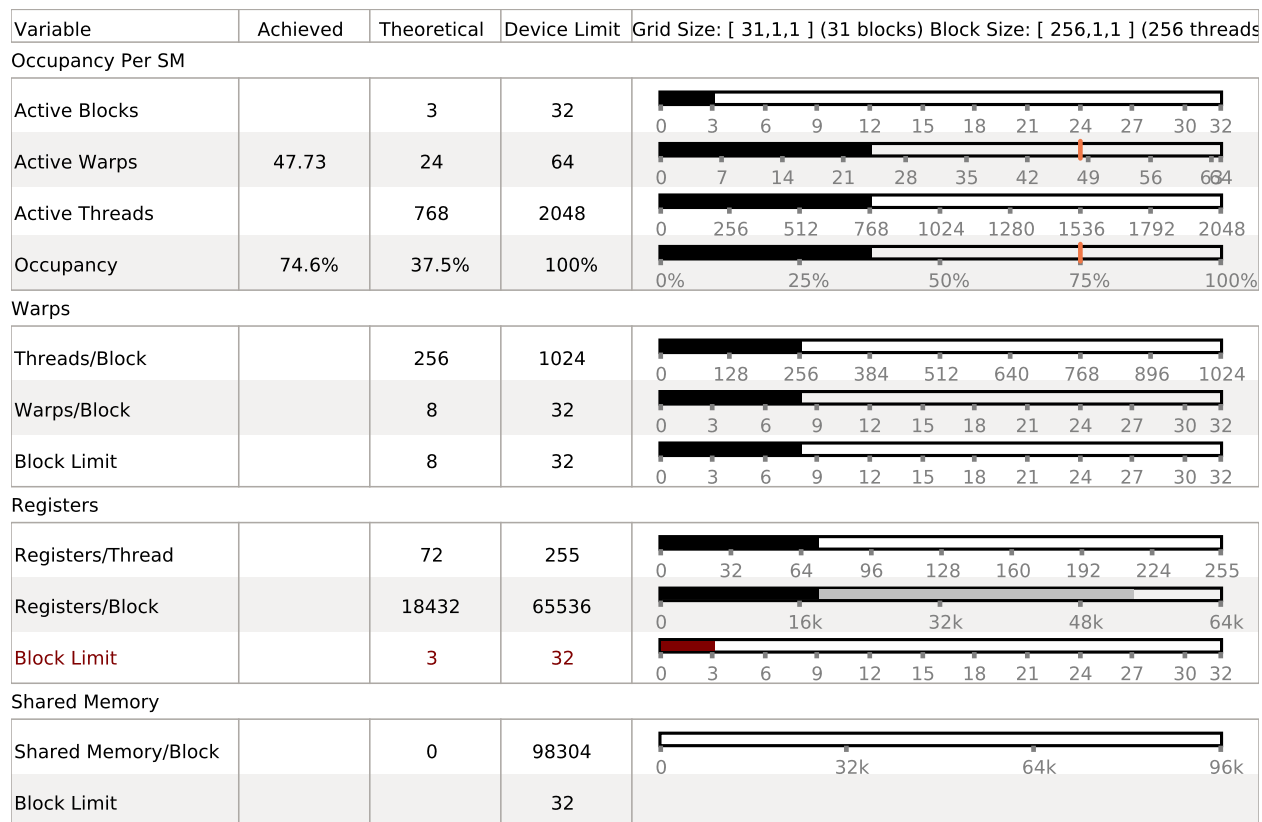
Sample distribution



2.2. GPU Utilization Is Limited By Register Usage

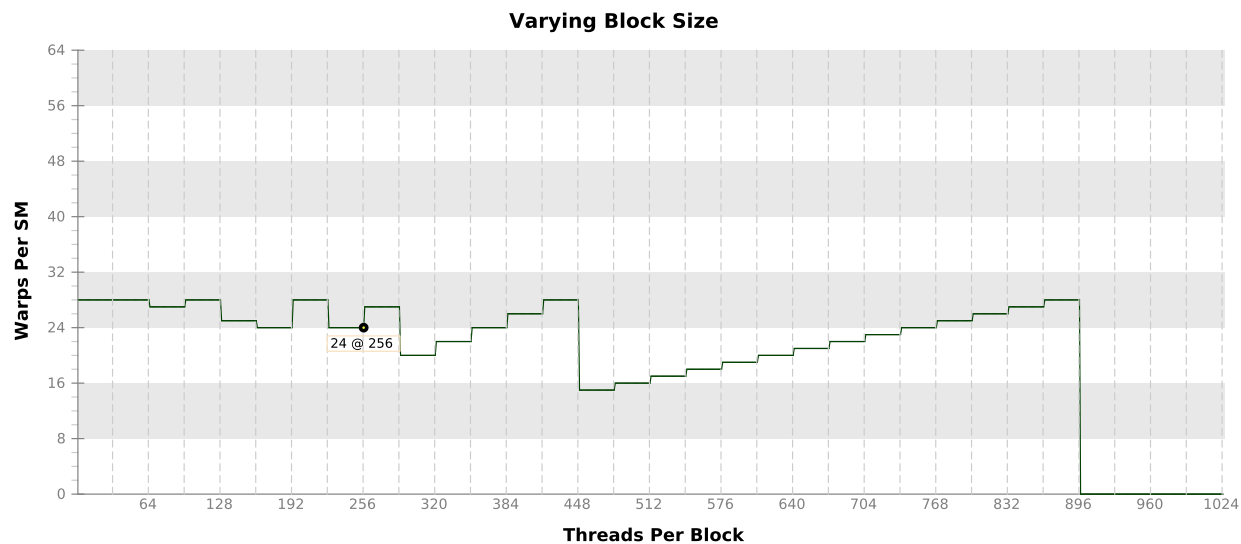
The kernel uses 72 registers for each thread (18432 registers for each block). This register usage is likely preventing the kernel from fully utilizing the GPU. Device "GeForce GTX 960" provides up to 65536 registers for each block. Because the kernel uses 18432 registers for each block each SM is limited to simultaneously executing 3 blocks (24 warps). Chart "Varying Register Count" below shows how changing register usage will change the number of blocks that can execute on each SM.

Optimization: Use the `-maxrregcount` flag or the `__launch_bounds__` qualifier to decrease the number of registers used by each thread. This will increase the number of blocks that can execute on each SM. On devices with Compute Capability 5.2 turning global cache off can increase the occupancy limited by register usage.

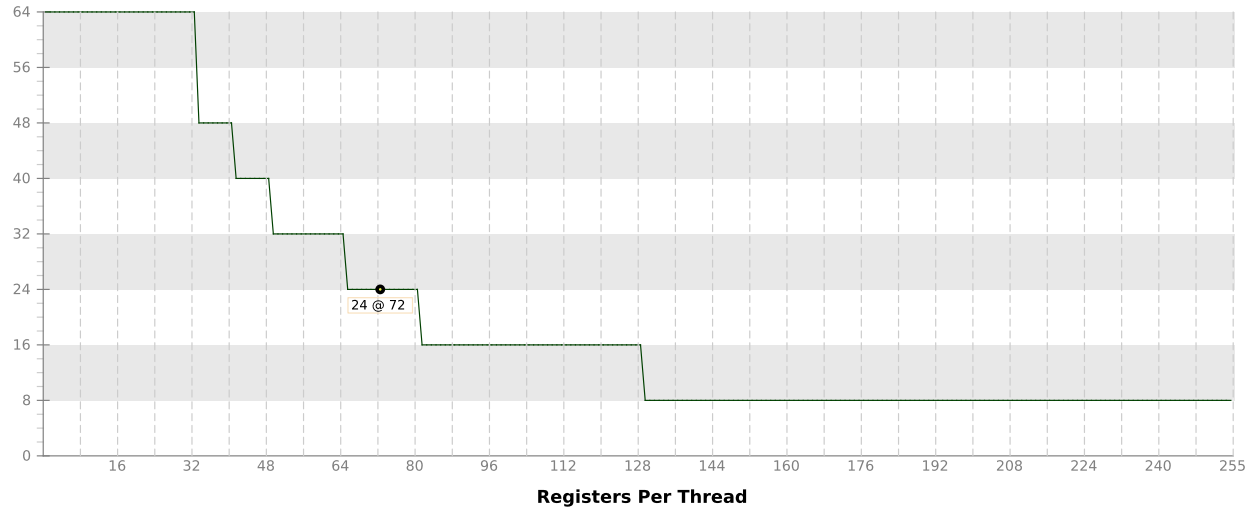


2.3. Occupancy Charts

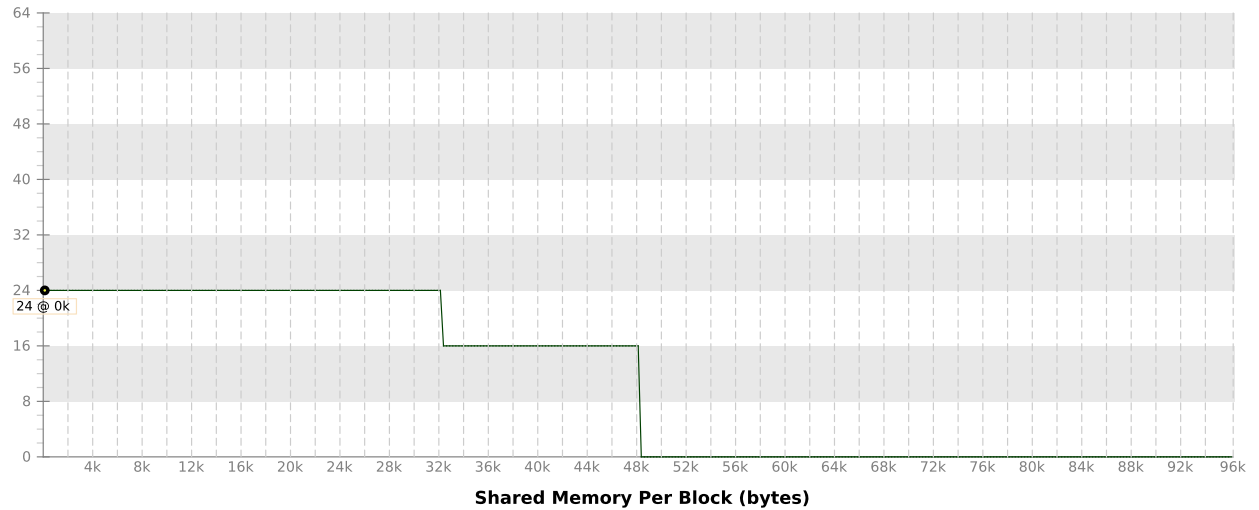
The following charts show how varying different components of the kernel will impact theoretical occupancy.



Varying Register Count



Varying Shared Memory Usage



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 Functions :

```
void computeHOGlocal<float, float, float, int=8, int=8, int=16, int=16, int=64>(float*, float*, float*, float*, int, int, int, int)
```

Maximum instruction execution count in assembly: 497664

Average instruction execution count in assembly: 27415

Instructions executed for the kernel: 54281902

Thread instructions executed for the kernel: 1736514629

Non-predicated thread instructions executed for the kernel: 1515530572

Warp non-predicated execution efficiency of the kernel: 87.2%

Warp execution efficiency of the kernel: 100.0%

Source files :

```
/home/adas/cuda-workspace/CudaVisionSysDeploy/Release/./src/init/./device/HOG/normalizeDescriptor.cuh  
/home/adas/cuda-workspace/CudaVisionSysDeploy/Release/./src/init/./device/HOG/HOGdescriptor.cuh  
/home/adas/cuda-workspace/CudaVisionSysDeploy/Release/./src/init/./device/HOG/addToHistogram.cuh  
/usr/local/cuda-7.5/targets/x86_64-linux/include/device_functions.hpp
```

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.

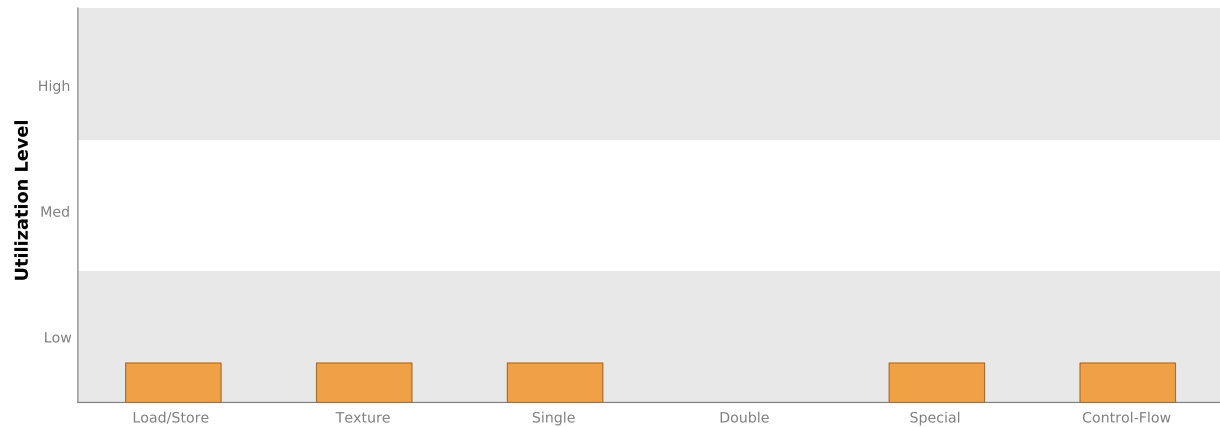
Texture - Load and store instructions for local, global, and texture memory.

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

Double - Double-precision floating-point arithmetic instructions.

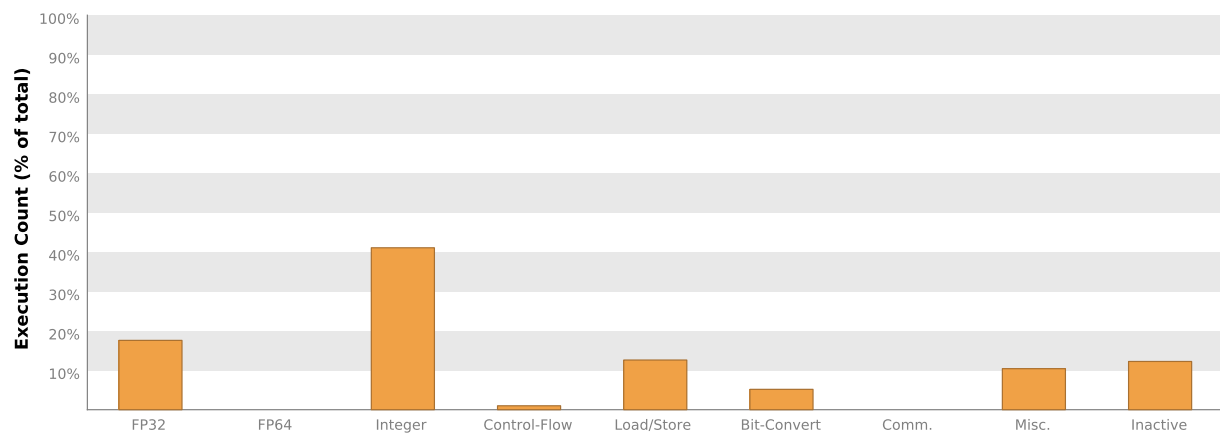
Special - Special arithmetic instructions such as sin, cos, popc, etc.

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



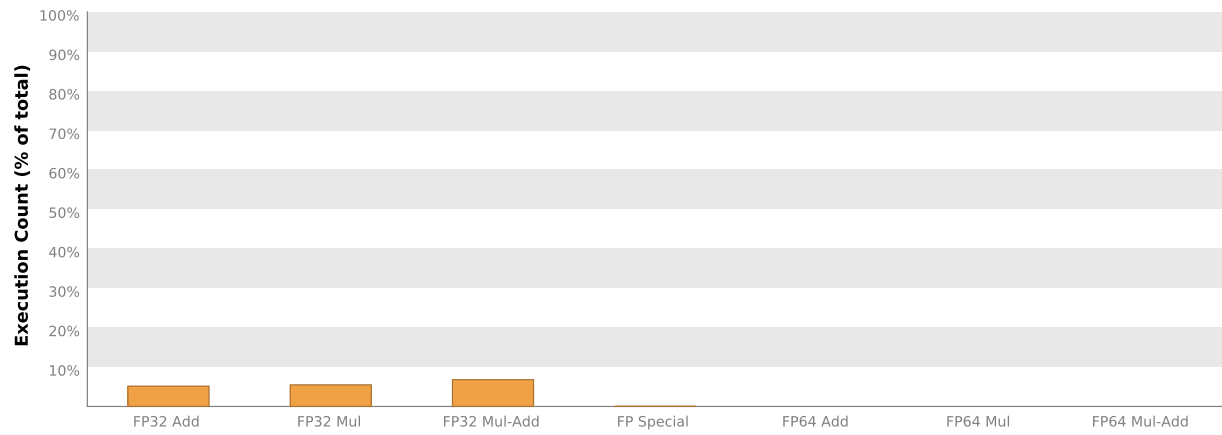
3.3. Instruction Execution Counts

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 | 0 | 0 B/s | |
| Shared Total | 0 | 0 B/s | |
| L2 Cache | | | |
| Reads | 110730252 | 33.137 GB/s | |
| Writes | 76141374 | 22.786 GB/s | |
| Total | 186871626 | 55.923 GB/s | |
| Unified Cache | | | |
| Local Loads | 0 | 0 B/s | |
| Local Stores | 0 | 0 B/s | |
| Global Loads | 112221460 | 33.137 GB/s | |
| Global Stores | 76141368 | 22.786 GB/s | |
| Texture Reads | 15769728 | 4.719 GB/s | |
| Unified Total | 204132556 | 60.642 GB/s | |
| Device Memory | | | |
| Reads | 93071361 | 27.853 GB/s | |
| Writes | 64728236 | 19.371 GB/s | |
| Total | 157799597 | 47.223 GB/s | |
| System Memory | | | |
| [PCIe configuration: Gen2 x16, 5 Gbit/s] | | | |
| Reads | 0 | 0 B/s | |
| Writes | 5 | 1.496 kB/s | |