ECE 408 / CS 483 Project
Milestone 2
Zigeng Zhu (zigeng2)

1. RAI Output

```
Dynamic Rate Limit: 30s
✱ Checking your authentication credentials.
✱ Preparing your project directory for upload.
✱ Uploading your project directory. This may take a few minutes.
 245.97 KiB / 245.97 KiB [████████████████████████████████████████████████████████] 100.00% 1.06 MiB/s 0s
✱ Folder uploaded. Server is now processing your submission.
✱ Your job request has been posted to the queue.
✱ Server has accepted your job submission and started to configure the container.
✱ Downloading your code.
✱ Using jnativ/ece408_minidnn_docker_sp21:latest as container image.
✱ Starting container.
✱ Running /bin/bash -c "mkdir /build/student_code && cp -rv /src/* /build/student_code"
'/src/cmake-build-debug' -> '/build/student_code/cmake-build-debug'
'/src/cmake-build-debug/CMakeCache.txt' -> '/build/student_code/cmake-build-debug/CMakeCache.txt'
'/src/cmake-build-debug/CMakeFiles' -> '/build/student_code/cmake-build-debug/CMakeFiles'
'/src/cmake-build-debug/CMakeFiles/3.17.3' -> '/build/student_code/cmake-build-debug/CMakeFiles/3.17.3'
'/src/cmake-build-debug/CMakeFiles/3.17.3/CMakeCCompiler.cmake' -> '/build/student_code/cmake-build-debug/CMakeFiles/3.17.3/CMakeCCompiler.cmake'
'/src/cmake-build-debug/CMakeFiles/3.17.3/CMakeCXXCompiler.cmake' -> '/build/student_code/cmake-build-debug/CMakeFiles/3.17.3/CMakeCXXCompiler.cmake'
'/src/cmake-build-debug/CMakeFiles/3.17.3/CMakeDetermineCompilerABI_C.bin' -> '/build/student_code/cmake-build-
debug/CMakeFiles/3.17.3/CMakeDetermineCompilerABI_C.bin'
'/src/cmake-build-debug/CMakeFiles/3.17.3/CMakeDetermineCompilerABI_CXX.bin' -> '/build/student_code/cmake-build-
debug/CMakeFiles/3.17.3/CMakeDetermineCompilerABI_CXX.bin'
'/src/cmake-build-debug/CMakeFiles/3.17.3/CMakeRCCompiler.cmake' -> '/build/student_code/cmake-build-debug/CMakeFiles/3.17.3/CMakeRCCompiler.cmake'
'/src/cmake-build-debug/CMakeFiles/3.17.3/CMakeSystem.cmake' -> '/build/student_code/cmake-build-debug/CMakeFiles/3.17.3/CMakeSystem.cmake'
'/src/cmake-build-debug/CMakeFiles/3.17.3/CompilerIdC' -> '/build/student_code/cmake-build-debug/CMakeFiles/3.17.3/CompilerIdC'
'/src/cmake-build-debug/CMakeFiles/3.17.3/CompilerIdC/CMakeCCompilerId.c' -> '/build/student_code/cmake-build-
debug/CMakeFiles/3.17.3/CompilerIdC/CMakeCCompilerId.c'
'/src/cmake-build-debug/CMakeFiles/3.17.3/CompilerIdC/a.exe' -> '/build/student_code/cmake-build-debug/CMakeFiles/3.17.3/CompilerIdC/a.exe'
'/src/cmake-build-debug/CMakeFiles/3.17.3/CompilerIdC/tmp' -> '/build/student_code/cmake-build-debug/CMakeFiles/3.17.3/CompilerIdC/tmp'
'/src/cmake-build-debug/CMakeFiles/3.17.3/CompilerIdCUDA' -> '/build/student_code/cmake-build-debug/CMakeFiles/3.17.3/CompilerIdCUDA'
'/src/cmake-build-debug/CMakeFiles/3.17.3/CompilerIdCUDA/CMakeCUDACompilerId.cu' -> '/build/student_code/cmake-build-
debug/CMakeFiles/3.17.3/CompilerIdCUDA/CMakeCUDACompilerId.cu'
'/src/cmake-build-debug/CMakeFiles/3.17.3/CompilerIdCUDA/tmp' -> '/build/student_code/cmake-build-debug/CMakeFiles/3.17.3/CompilerIdCUDA/tmp'
'/src/cmake-build-debug/CMakeFiles/3.17.3/CompilerIdCXX' -> '/build/student_code/cmake-build-debug/CMakeFiles/3.17.3/CompilerIdCXX'
'/src/cmake-build-debug/CMakeFiles/3.17.3/CompilerIdCXX/CMakeCXXCompilerId.cpp' -> '/build/student_code/cmake-build-
debug/CMakeFiles/3.17.3/CompilerIdCXX/CMakeCXXCompilerId.cpp'
'/src/cmake-build-debug/CMakeFiles/3.17.3/CompilerIdCXX/a.exe' -> '/build/student_code/cmake-build-debug/CMakeFiles/3.17.3/CompilerIdCXX/a.exe'
'/src/cmake-build-debug/CMakeFiles/3.17.3/CompilerIdCXX/tmp' -> '/build/student_code/cmake-build-debug/CMakeFiles/3.17.3/CompilerIdCXX/tmp'
'/src/cmake-build-debug/CMakeFiles/CMakeDirectoryInformation.cmake' -> '/build/student_code/cmake-build-debug/CMakeFiles/CMakeDirectoryInformation.cmake'
'/src/cmake-build-debug/CMakeFiles/CMakeError.log' -> '/build/student_code/cmake-build-debug/CMakeFiles/CMakeError.log'
'/src/cmake-build-debug/CMakeFiles/CMakeOutput.log' -> '/build/student_code/cmake-build-debug/CMakeFiles/CMakeOutput.log'
'/src/cmake-build-debug/CMakeFiles/CMakeTmp' -> '/build/student_code/cmake-build-debug/CMakeFiles/CMakeTmp'
'/src/cmake-build-debug/CMakeFiles/Makefile.cmake' -> '/build/student_code/cmake-build-debug/CMakeFiles/Makefile.cmake'
'/src/cmake-build-debug/CMakeFiles/Makefile2' -> '/build/student_code/cmake-build-debug/CMakeFiles/Makefile2'
'/src/cmake-build-debug/CMakeFiles/TargetDirectories.txt' -> '/build/student_code/cmake-build-debug/CMakeFiles/TargetDirectories.txt'
'/src/cmake-build-debug/CMakeFiles/clion-environment.txt' -> '/build/student_code/cmake-build-debug/CMakeFiles/clion-environment.txt'
'/src/cmake-build-debug/CMakeFiles/clion-log.txt' -> '/build/student_code/cmake-build-debug/CMakeFiles/clion-log.txt'
'/src/cmake-build-debug/CMakeFiles/cmake.check_cache' -> '/build/student_code/cmake-build-debug/CMakeFiles/cmake.check_cache'
'/src/cmake-build-debug/CMakeFiles/ece408_project.dir' -> '/build/student_code/cmak
e-build-debug/CMakeFiles/ece408_project.dir'
'/src/cmake-build-debug/CMakeFiles/ece408_project.dir/DependInfo.cmake' -> '/build/student_code/cmake-build-
debug/CMakeFiles/ece408_project.dir/DependInfo.cmake'
'/src/cmake-build-debug/CMakeFiles/ece408_project.dir/build.make' -> '/build/student_code/cmake-build-debug/CMakeFiles/ece408_project.dir/build.make'
'/src/cmake-build-debug/CMakeFiles/ece408_project.dir/cmake_clean.cmake' -> '/build/student_code/cmake-build-
debug/CMakeFiles/ece408_project.dir/cmake_clean.cmake'
'/src/cmake-build-debug/CMakeFiles/ece408_project.dir/custom' -> '/build/student_code/cmake-build-debug/CMakeFiles/ece408_project.dir/custom'
'/src/cmake-build-debug/CMakeFiles/ece408_project.dir/depend.make' -> '/build/student_code/cmake-build-debug/CMakeFiles/ece408_project.dir/depend.make'
'/src/cmake-build-debug/CMakeFiles/ece408_project.dir/flags.make' -> '/build/student_code/cmake-build-debug/CMakeFiles/ece408_project.dir/flags.make'
'/src/cmake-build-debug/CMakeFiles/ece408_project.dir/link.txt' -> '/build/student_code/cmake-build-debug/CMakeFiles/ece408_project.dir/link.txt'
'/src/cmake-build-debug/CMakeFiles/ece408_project.dir/progress.make' -> '/build/student_code/cmake-build-debug/CMakeFiles/ece408_project.dir/progress.make'
'/src/cmake-build-debug/CMakeFiles/progress.marks' -> '/build/student_code/cmake-build-debug/CMakeFiles/progress.marks'
'/src/cmake-build-debug/Makefile' -> '/build/student_code/cmake-build-debug/Makefile'
'/src/cmake-build-debug/cmake_install.cmake' -> '/build/student_code/cmake-build-debug/cmake_install.cmake'
'/src/cmake-build-debug/ece408_project.cbp' -> '/build/student_code/cmake-build-debug/ece408_project.cbp'
'/src/custom' -> '/build/student_code/custom'
'/src/custom/cpu-new-forward.cc' -> '/build/student_code/custom/cpu-new-forward.cc'
'/src/custom/cpu-new-forward.h' -> '/build/student_code/custom/cpu-new-forward.h'
'/src/custom/gpu-new-forward.h' -> '/build/student_code/custom/gpu-new-forward.h'
'/src/custom/new-forward.cu' -> '/build/student_code/custom/new-forward.cu'
'/src/final.cc' -> '/build/student_code/final.cc'
'/src/m1.cc' -> '/build/student_code/m1.cc'
'/src/m2.cc' -> '/build/student_code/m2.cc'
'/src/m3.cc' -> '/build/student_code/m3.cc'
'/src/rai_build.yml' -> '/build/student_code/rai_build.yml'
'/src/readme.md' -> '/build/student_code/readme.md'
'/src/report.txt' -> '/build/student_code/report.txt'
✱ Running /bin/bash -c "cp /ece408/project/build/weights-86.bin /build"
✱ Running /bin/bash -c "cp -rv /src/custom/* /ece408/project/src/layer/custom"
'/src/custom/cpu-new-forward.cc' -> '/ece408/project/src/layer/custom/cpu-new-forward.cc'
'/src/custom/cpu-new-forward.h' -> '/ece408/project/src/layer/custom/cpu-new-forward.h'
'/src/custom/gpu-new-forward.h' -> '/ece408/project/src/layer/custom/gpu-new-forward.h'
'/src/custom/new-forward.cu' -> '/ece408/project/src/layer/custom/new-forward.cu'
✱ Running /bin/bash -c "cmake  /ece408/project/ && make -j8"
-- The C compiler identification is GNU 7.5.0
-- The CXX compiler identification is GNU 7.5.0
-- Check for working C compiler: /usr/bin/cc
-- Check for working C compiler: /usr/bin/cc -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Detecting C compile features
-- Detecting C compile features - done
-- Check for working CXX compiler: /usr/bin/c++
-- Check for working CXX compiler: /usr/bin/c++ -- works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Looking for pthread.h
-- Looking for pthread.h - found
```

```
-- Looking for pthread_create
-- Looking for pthread_create - not found
-- Looking for pthread_create in pthreads
-- Looking for pthread_create in pthreads - not found
-- Looking for pthread_create in pthread
-- Looking for pthread_create in pthread - found
-- Found Threads: TRUE
-- Found CUDA: /usr/local/cuda (found version "10.2")
-- Configuring done
-- Generating done
-- Build files have been written to: /build
[  3%] Building NVCC (Device) object src/CMakeFiles/GpuConv.dir/layer/custom/GpuConv_generated_new-forward.cu.o
Scanning dependencies of target ece408net
[  6%] Building NVCC (Device) object src/CMakeFiles/GpuConv.dir/layer/custom/GpuConv_generated_gpu-utils.cu.o
[ 10%] Building CXX object CMakeFiles/ece408net.dir/ece408net.cc.o
[ 13%] Linking CXX static library libece408net.a
[ 13%] Built target ece408net
Scanning dependencies of target GpuConv
[ 17%] Linking CXX static library libGpuConv.a
[ 17%] Built target GpuConv
Scanning dependencies of target MiniDNNLib
[ 20%] Building CXX object src/CMakeFiles/MiniDNNLib.dir/layer/conv.cc.o
[ 24%] Building CXX object src/CMakeFiles/MiniDNNLib.dir/mnist.cc.o
[ 27%] Building CXX object src/CMakeFiles/MiniDNNLib.dir/layer/max_pooling.cc.o
[ 31%] Building CXX object src/CMakeFiles/MiniDNNLib.dir/layer/fully_connected.cc.o
[ 34%] Building CXX object src/CMakeFiles/MiniDNNLib.dir/layer/conv_cust.cc.o
[ 37%] Building CXX object src/CMakeFiles/MiniDNNLib.dir/network.cc.o
[ 41%] Building CXX object src/CMakeFiles/MiniDNNLib.dir/layer/ave_pooling.cc.o
[ 44%] Building CXX object src/CMakeFiles/MiniDNNLib.dir/layer/conv_cpu.cc.o
[ 48%] Building CXX object src/CMakeFiles/MiniDNNLib.dir/layer/relu.cc.o
[ 51%] Building CXX object src/CMakeFiles/MiniDNNLib.dir/layer/sigmoid.cc.o
[ 55%] Building CXX object src/CMakeFiles/MiniDNNLib.dir/layer/softmax.cc.o
[ 58%] Building CXX object src/CMakeFiles/MiniDNNLib.dir/loss/cross_entropy_loss.cc.o
[ 62%] Building CXX object src/CMakeFiles/MiniDNNLib.dir/loss/mse_loss.cc.o
[ 65%] Building CXX object src/CMakeFiles/MiniDNNLib.dir/optimizer/sgd.cc.o
[ 68%] Building CXX object src/CMakeFiles/MiniDNNLib.dir/layer/custom/cpu-new-forward.cc.o
[ 72%] Linking CXX static library libMiniDNNLib.a
[ 72%] Built target MiniDNNLib
Scanning dependencies of target final
Scanning dependencies of target m2
Scanning dependencies of target m1
Scanning dependencies of target m3
[ 75%] Building CXX object CMakeFiles/final.dir/final.cc.o
[ 79%] Building CXX object CMakeFiles/m1.dir/m1.cc.o
[ 82%] Building CXX object CMakeFiles/m3.dir/m3.cc.o
[ 86%] Building CXX object CMakeFiles/m2.dir/m2.cc.o
[ 89%] Linking CXX executable final
[ 93%] Linking CXX executable m1
[ 96%] Linking CXX executable m3
[100%] Linking CXX executable m2
[100%] Built target m1
[100%] Built target m3
[100%] Built target final
[100%] Built target m2
* Running bash -c "cuda-memcheck ./m2"   \\ Output will appear after run is complete.


Test batch size: 10000
Loading fashion-mnist data...Done
Loading model...Done
Conv-GPU==
Op Time: 15.9956 ms
Conv-GPU==
Op Time: 62.2208 ms

Test Accuracy: 0.8714


Test batch size: 1000
Loading fashion-mnist data...Done
Loading model...Done
Conv-GPU==
Op Time: 1.62294 ms
Conv-GPU==
Op Time: 6.25213 ms

Test Accuracy: 0.886


Test batch size: 100
Loading fashion-mnist data...Done
Loading model...Done
Conv-GPU==
Op Time: 0.174584 ms
Conv-GPU==
Op Time: 0.636023 ms

Test Accuracy: 0.86
```

2. Nsys Profiling

```
* Running profile --stats=true ./m2   \\ Output will appear after run is complete.
**** collection configuration ****
        force-overwrite = false
        stop-on-exit = true
        export_sqlite = true
        stats = true
        capture-range = none
        stop-on-range-end = false
        Beta: ftrace events:
        ftrace-keep-user-config = false
        trace-GPU-context-switch = false
        delay = 0 seconds
        duration = 0 seconds
        kill = signal number 15
        inherit-environment = true
        show-output = true
        trace-fork-before-exec = false
        sample_cpu = true
        backtrace_method = LBR
        wait = all
        trace_cublas = false
        trace_cuda = true
```

```
        trace_cudnn = false
        trace_nvtx = true
        trace_mpi = false
        trace_openacc = false
        trace_vulkan = false
        trace_opengl = true
        trace_osrt = true
        osrt-threshold = 0 nanoseconds
        cudabacktrace = false
        cudabacktrace-threshold = 0 nanoseconds
        profile_processes = tree
        application command = ./m2
        application arguments =
        application working directory = /build
        NVTX profiler range trigger =
        NVTX profiler domain trigger =
        environment variables:
        Collecting data...
Test batch size: 10000
Loading fashion-mnist data...Done
Loading model...Done
Conv-GPU==
Op Time: 16.1061 ms
Conv-GPU==
Op Time: 62.5243 ms

Test Accuracy: 0.8714

        Generating the /build/report1.qdstrm file.
        Capturing raw events...

        **** WARNING: The collection generated 649332 total events. ****
        Importing this QDSTRM file into the NVIDIA Nsight Systems GUI may take several minutes to complete.

        Capturing symbol files...
        Saving diagnostics...
        Saving qdstrm file to disk...
        Finished saving file.


Importing the qdstrm file using /opt/nvidia/nsight-systems/2019.5.2/host-linux-x64/QdstrmImporter.

Importing...

Importing [==================================================100%]
Saving report to file "/build/report1.qdrep"
Report file saved.
Please discard the qdstrm file and use the qdrep file instead.

Removed /build/report1.qdstrm as it was successfully imported.
Please use the qdrep file instead.

Exporting the qdrep file to SQLite database using /opt/nvidia/nsight-systems/2019.5.2/host-linux-x64/nsys-exporter.

Exporting 649235 events:

0%   10   20   30   40   50   60   70   80   90   100%
|----|----|----|----|----|----|----|----|----|----|
**************************************************

Exported successfully to
/build/report1.sqlite

Generating CUDA API Statistics...
```

CUDA API Statistics (nanoseconds)

Time(%)	Total Time	Calls	Average	Minimum	Maximum	Name
76.4	926301586	8	115787698.3	16871	480928403	cudaMemcpy
15.8	191541403	8	23942675.4	71995	186841612	cudaMalloc
6.5	78607598	6	13101266.3	3396	62497948	cudaDeviceSynchronize
1.1	13329591	6	2221598.5	16936	13217962	cudaLaunchKernel
0.2	2501080	8	312635.0	56141	700934	cudaFree

Generating CUDA Kernel Statistics...

Generating CUDA Memory Operation Statistics...

CUDA Kernel Statistics (nanoseconds)

Time(%)	Total Time	Instances	Average	Minimum	Maximum	Name
100.0	78581833	2	39290916.5	16085106	62496727	conv_forward_kernel
0.0	2720	2	1360.0	1312	1408	prefn_marker_kernel
0.0	2528	2	1264.0	1248	1280	do_not_remove_this_kernel

CUDA Memory Operation Statistics (nanoseconds)

Time(%)	Total Time	Operations	Average	Minimum	Maximum	Name
90.2	823781659	2	411890829.5	343545631	480236028	[CUDA memcpy DtoH]
9.8	89528928	6	14921488.0	1216	47995098	[CUDA memcpy HtoD]

CUDA Memory Operation Statistics (KiB)

Total	Operations	Average	Minimum	Maximum	Name
1722500.0	2	861250.0	722500.000	1000000.0	[CUDA memcpy DtoH]
538919.0	6	89819.0	0.004	288906.0	[CUDA memcpy HtoD]

Generating Operating System Runtime API Statistics...
Operating System Runtime API Statistics (nanoseconds)

Time(%)	Total Time	Calls	Average	Minimum	Maximum	Name
33.3	85706389293	871	98399987.7	21151	100219334	sem_timedwait
33.3	85672901853	870	98474599.8	32784	100251650	poll
21.1	54148807826	2	27074403913.0	19624237671	34524570155	pthread_cond_wait
12.3	31508969399	63	500142371.4	500103957	500164765	pthread_cond_timedwait
0.1	129406343	905	142990.4	1003	17096414	ioctl
0.0	19876216	26	764469.8	1250	19797135	fopen
0.0	16341310	9072	1801.3	1029	18123	read
0.0	2797174	97	28836.8	1008	1134607	mmap
0.0	1229095	101	12169.3	5195	27632	open64
0.0	316699	5	63339.8	38352	117225	pthread_create
0.0	170063	3	56687.7	53018	61831	fgets
0.0	80097	1	80097.0	80097	80097	pthread_mutex_lock
0.0	79587	18	4421.5	1182	14546	munmap
0.0	62591	15	4172.7	2190	6604	write
0.0	37715	7	5387.9	3589	6613	fflush
0.0	30620	5	6124.0	2848	9383	open
0.0	30130	3	10043.3	8154	13471	fopen64
0.0	26318	10	2631.8	1054	7484	fclose
0.0	11730	2	5865.0	5050	6680	socket
0.0	9260	2	4630.0	4085	5175	pthread_cond_signal
0.0	8155	1	8155.0	8155	8155	connect
0.0	6397	1	6397.0	6397	6397	pipe2
0.0	2598	1	2598.0	2598	2598	fwrite
0.0	1469	1	1469.0	1469	1469	bind
0.0	1087	1	1087.0	1087	1087	fcntl

Generating NVTX Push-Pop Range Statistics...
NVTX Push-Pop Range Statistics (nanoseconds)

3. **Difference between kernels and API calls**
   API calls are pre-implemented functions (in CUDA libraries) that are called in the host to interact with the device (cudaMemcpy etc.). Kernels are function that is being executed on the GPU.

4. **Nsight Compute Screenshot (batch size = 10000)**

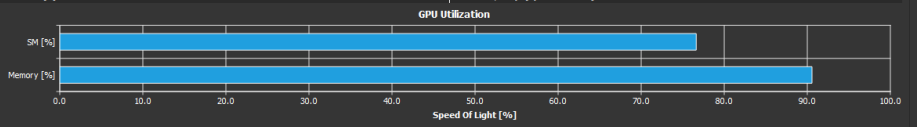
Page: Details | Launch: 1 - 122 - conv_forward_kernel | Add Baseline | Apply Rules | Save as Image

Current 122 - conv_forward_kernel (10000, 4, 25)x(16, 1... **Time:** 16.07 msecond **Cycles:** 19,283,167 **Regs:** 32 **GPU:** TITAN V **SM Frequency:** 1.20 cycle/nsecond **CC:** 7.0 **Process:** [557] m2

GPU Speed Of Light

High-level overview of the utilization for compute and memory resources of the GPU. For each unit, the Speed Of Light (SOL) reports the achieved percentage of utilization with respect to the theoretical maximum. High-level overview of the utilization for compute and memory resources of the GPU presented as a roofline chart.

SOL SM [%]	76.63	Duration [msecond]	16.07
SOL Memory [%]	90.57	Elapsed Cycles [cycle]	19,283,167
SOL L1/TEX Cache [%]	90.59	SM Active Cycles [cycle]	19,278,233.91
SOL L2 Cache [%]	12.43	SM Frequency [cycle/nsecond]	1.20
SOL DRAM [%]	39.93	DRAM Frequency [cycle/usecond]	850.43

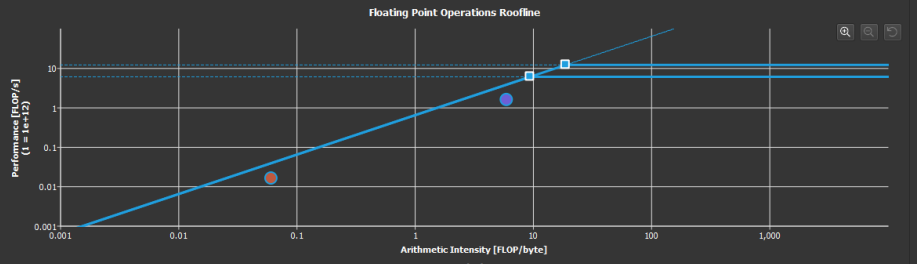


SOL SM Breakdown

SOL SM: Issue Active [%]	76.63
SOL SM: Inst Executed [%]	76.62
SOL SM: Inst Executed Pipe Lsu [%]	57.05
SOL SM: Pipe Alu Cycles Active [%]	55.49
SOL SM: Pipe Fma Cycles Active [%]	49.27
SOL SM: Mio Inst Issued [%]	29.30
SOL SM: Mio2rf Writeback Active [%]	28.41
SOL SM: Inst Executed Pipe Cbu Pred On Any [%]	18.15
SOL SM: Inst Executed Pipe Xu [%]	5.19
SOL SM: Inst Executed Pipe Adu [%]	3.11
SOL SM: Mio Pq Read Cycles Active [%]	0.61
SOL SM: Mio Pq Write Cycles Active [%]	0.52
SOL SM: Pipe Fp64 Cycles Active [%]	0.52
SOL SM: Pipe Shared Cycles Active [%]	0.52
SOL IDC: Request Cycles Active [%]	0
SOL SM: Inst Executed Pipe Fp16 [%]	0
SOL SM: Inst Executed Pipe Ipa [%]	0
SOL SM: Inst Executed Pipe Tex [%]	0
SOL SM: Pipe Tensor Cycles Active [%]	0

SOL Memory Breakdown

SOL L1: Data Pipe Lsu Wavefronts [%]	90.57
SOL L1: Lsu Writeback Active [%]	68.42
SOL L1: Lsuin Requests [%]	57.05
SOL GPU: Dram Throughput [%]	39.93
SOL L1: Data Bank Reads [%]	13.36
SOL L2: T Sectors [%]	12.43
SOL L2: D Sectors Fill Device [%]	11.26
SOL L2: Xbar2lts Cycles Active [%]	9.13
SOL L2: Lts2xbar Cycles Active [%]	8.79
SOL L2: T Tag Requests [%]	7.32
SOL L2: D Sectors [%]	6.83
SOL L1: M L1tex2xbar Req Cycles Active [%]	5.21
SOL L1: M Xbar2l1tex Read Sectors [%]	5.01
SOL L1: Data Bank Writes [%]	0.89
SOL L1: F Wavefronts [%]	0.00
SOL L1: Texin Sm2tex Req Cycles Active [%]	0.00
SOL L1: Data Pipe Tex Wavefronts [%]	0
SOL L1: Tex Writeback Active [%]	0
SOL L2: D Atomic Input Cycles Active [%]	0
SOL L2: D Sectors Fill Sysmem [%]	0



Recommendations

Bottleneck The kernel is utilizing greater than 80.0% of the available compute or memory performance of the device. To further improve performance, work will likely need to be shifted from the most utilized to another unit. Start by analyzing workloads in the Memory Workload Analysis section.

Roofline Analysis The ratio of peak float (fp32) to double (fp64) performance on this device is 2:1. The kernel achieved 13% of this device's fp32 peak performance and close to 0% of its fp64 peak performance.

Compute Workload Analysis

Detailed analysis of the compute resources of the streaming multiprocessors (SM), including the achieved instructions per clock (IPC) and the utilization of each available pipeline. Pipelines with very high utilization might limit the overall performance.

Executed Ipc Elapsed [inst/cycle]	3.86	SM Busy [%]	76.65
Executed Ipc Active [inst/cycle]	3.87	Issue Slots Busy [%]	76.65
Issued Ipc Active [inst/cycle]	3.87		

Memory Workload Analysis

Detailed analysis of the memory resources of the GPU. Memory can become a limiting factor for the overall kernel performance when fully utilizing the involved hardware units (Mem Busy), exhausting the available communication bandwidth between those units (Max Bandwidth), or by reaching the maximum throughput of issuing memory instructions (Mem Pipes Busy). Detailed chart of the memory units. Detailed tables with data for each memory unit. Deprecated UI elements for backwards compatibility.

Memory Throughput [Gbyte/second]	260.80	Mem Busy [%]	90.57
L1/TEX Hit Rate [%]	95.90	Max Bandwidth [%]	68.42
L2 Hit Rate [%]	37.10	Mem Pipes Busy [%]	57.05

Scheduler Statistics

Summary of the activity of the schedulers issuing instructions. Each scheduler maintains a pool of warps that it can issue instructions for. The upper bound of warps in the pool (Theoretical Warps) is limited by the launch configuration. On every cycle each scheduler checks the state of the allocated warps in the pool (Active Warps). Active warps that are not stalled (Eligible Warps) are ready to issue their next instruction. From the set of eligible warps the scheduler selects a single warp from which to issue one or more instructions (Issued Warp). On cycles with no eligible warps, the issue slot is skipped and no instruction is issued. Having many skipped issue slots indicates poor latency hiding.

Active Warps Per Scheduler [warp]	13.83	No Eligible [%]	23.35
Eligible Warps Per Scheduler [warp]	4.00	One or More Eligible [%]	76.65
Issued Warp Per Scheduler	0.77		

Warp State Statistics

Analysis of the states in which all warps spent cycles during the kernel execution. The warp states describe a warp's readiness or inability to issue its next instruction. The warp cycles per instruction define the latency between two consecutive instructions. The higher the value, the more warp parallelism is required to hide this latency. For each warp state, the chart shows the average number of cycles spent in that state per issued instruction. Stalls are not always impacting the overall performance nor are they completely avoidable. Only focus on stall reasons if the schedulers fail to issue every cycle. When executing a kernel with mixed library and user code, these metrics show the combined values.

Warp Cycles Per Issued Instruction [cycle]	18.05	Avg. Active Threads Per Warp	32
Warp Cycles Per Executed Instruction [cycle]	18.05	Avg. Not Predicated Off Threads Per Warp	29.15

Instruction Statistics

Statistics of the executed low-level assembly instructions (SASS). The instruction mix provides insight into the types and frequency of the executed instructions. A narrow mix of instruction types implies a dependency on few instruction pipelines, while others remain unused. Using multiple pipelines allows hiding latencies and enables parallel execution. Note that 'Instructions/Opcode' and 'Executed Instructions' are measured differently and can diverge if cycles are spent in system calls.

Executed Instructions [inst]	4,728,000,000	Avg. Executed Instructions Per Scheduler [inst]	14,775,000
Issued Instructions [inst]	4,728,314,126	Avg. Issued Instructions Per Scheduler [inst]	14,775,981.64

Launch Statistics

Summary of the configuration used to launch the kernel. The launch configuration defines the size of the kernel grid, the division of the grid into blocks, and the GPU resources needed to execute the kernel. Choosing an efficient launch configuration maximizes device utilization.

Grid Size	1,000,000	Registers Per Thread [register/thread]	32
Block Size	256	Static Shared Memory Per Block [byte/block]	0
Threads [thread]	256,000,000	Dynamic Shared Memory Per Block [byte/block]	0
Waves Per SM	1,562.50	Driver Shared Memory Per Block [byte/block]	0
		Shared Memory Configuration Size [byte]	0

Occupancy

Occupancy is the ratio of the number of active warps per multiprocessor to the maximum number of possible active warps. Another way to view occupancy is the percentage of the hardware's ability to process warps that is actively in use. Higher occupancy does not always result in higher performance, however, low occupancy always reduces the ability to hide latencies, resulting in overall performance degradation. Large discrepancies between the theoretical and the achieved occupancy during execution typically indicates highly imbalanced workloads.

Theoretical Occupancy [%]	100	Block Limit Registers [block]	8
Theoretical Active Warps per SM [warp]	64	Block Limit Shared Mem [block]	32
Achieved Occupancy [%]	86.60	Block Limit Warps [block]	8
Achieved Active Warps Per SM [warp]	55.42	Block Limit SM [block]	32

Source Counters

Source metrics, including warp stall reasons. Sampling Data metrics are periodically sampled over the kernel runtime. They indicate when warps were stalled and couldn't be scheduled. See the documentation for a description of all stall reasons. Only focus on stalls if the schedulers fail to issue every cycle.

Page: Details | Launch: 4 - 143 - conv_forward_kernel | Add Baseline | Apply Rules | Save as Image

Current 143 - conv_forward_kernel (10000, 16, 9)x(16, 1... **Time:** 63.21 msecond **Cycles:** 75,842,128 **Regs:** 32 **GPU:** TITAN V **SM Frequency:** 1.20 cycle/nsecond **CC:** 7.0 **Process:** [557] m2

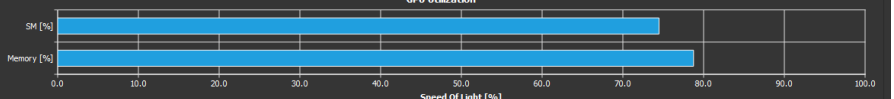
GPU Speed Of Light

All

High-level overview of the utilization for compute and memory resources of the GPU. For each unit, the Speed Of Light (SOL) reports the achieved percentage of utilization with respect to the theoretical maximum. High-level overview of the utilization for compute and memory resources of the GPU presented as a roofline chart.

SOL SM [%]	74.48	Duration [msecond]	63.21
SOL Memory [%]	78.76	Elapsed Cycles [cycle]	75,842,128
SOL L1/TEX Cache [%]	78.75	SM Active Cycles [cycle]	75,843,670.83
SOL L2 Cache [%]	9.56	SM Frequency [cycle/nsecond]	1.20
SOL DRAM [%]	28.54	DRAM Frequency [cycle/usecond]	850.42

GPU Utilization



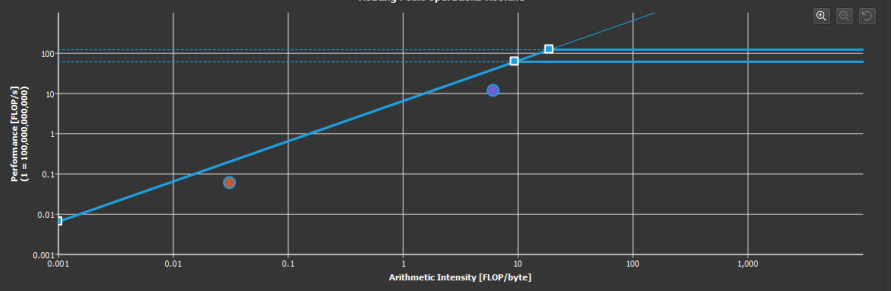
SOL SM Breakdown

SOL SM: Issue Active [%]	74.48
SOL SM: Inst Executed [%]	74.48
SOL SM: Inst Executed Pipe Lsu [%]	57.73
SOL SM: Pipe Alu Cycles Active [%]	52.52
SOL SM: Pipe Fma Cycles Active [%]	49.05
SOL SM: Mio Inst Issued [%]	29.09
SOL SM: Mio2rf Writeback Active [%]	27.93
SOL SM: Inst Executed Pipe Cbu Pred On Any [%]	18.93
SOL SM: Inst Executed Pipe Xu [%]	1.90
SOL SM: Inst Executed Pipe Adu [%]	0.92
SOL SM: Pipe Fp64 Cycles Active [%]	0.19
SOL SM: Pipe Shared Cycles Active [%]	0.19
SOL SM: Mio Pq Read Cycles Active [%]	0.15
SOL SM: Mio Pq Write Cycles Active [%]	0.13
SOL IDC: Request Cycles Active [%]	0
SOL SM: Inst Executed Pipe Fp16 [%]	0
SOL SM: Inst Executed Pipe Ipa [%]	0
SOL SM: Inst Executed Pipe Tex [%]	0
SOL SM: Pipe Tensor Cycles Active [%]	0

SOL Memory Breakdown

SOL L1: Data Pipe Lsu Wavefronts [%]	78.76
SOL L1: Lsu Writeback Active [%]	58.87
SOL L1: Lsuin Requests [%]	57.73
SOL GPU: Dram Throughput [%]	28.54
SOL L1: Data Bank Reads [%]	9.80
SOL L2: D Sectors Fill Device [%]	9.56
SOL L2: T Sectors [%]	7.73
SOL L2: Xbar2lts Cycles Active [%]	6.75
SOL L2: Lts2xbar Cycles Active [%]	6.71
SOL L2: T Tag Requests [%]	5.82
SOL L2: D Sectors [%]	4.58
SOL L1: M L1tex2xbar Req Cycles Active [%]	3.88
SOL L1: M Xbar2l1tex Read Sectors [%]	3.85
SOL L1: Data Bank Writes [%]	0.53
SOL L1: F Wavefronts [%]	0.00
SOL L1: Texin Sm2tex Req Cycles Active [%]	0.00
SOL L1: Data Pipe Tex Wavefronts [%]	0
SOL L1: Tex Writeback Active [%]	0
SOL L2: D Atomic Input Cycles Active [%]	0
SOL L2: D Sectors Fill Sysmem [%]	0

Floating Point Operations Roofline



Recommendations

Bottleneck Compute and Memory are well-balanced: To reduce runtime, both computation and memory traffic must be reduced. Check both the Compute Workload Analysis and Memory Workload Analysis report sections.

Roofline Analysis The ratio of peak float (fp32) to double (fp64) performance on this device is 2:1. The kernel achieved 9% of this device's fp32 peak performance and close to 0% of its fp64 peak performance.

Compute Workload Analysis

Detailed analysis of the compute resources of the streaming multiprocessors (SM), including the achieved instructions per clock (IPC) and the utilization of each available pipeline. Pipelines with very high utilization might limit the overall performance.

Executed Ipc Elapsed [inst/cycle]	2.98	SM Busy [%]	74.48
Executed Ipc Active [inst/cycle]	2.98	Issue Slots Busy [%]	74.48
Issued Ipc Active [inst/cycle]	2.98		

Memory Workload Analysis

All

Detailed analysis of the memory resources of the GPU. Memory can become a limiting factor for the overall kernel performance when fully utilizing the involved hardware units (Mem Busy), exhausting the available communication bandwidth between those units (Max Bandwidth), or by reaching the maximum throughput of issuing memory instructions (Mem Pipes Busy). Detailed chart of the memory units. Detailed tables with data for each memory unit. Deprecated UI elements for backwards compatibility.

Memory Throughput [Gbyte/second]	186.38	Mem Busy [%]	78.76
L1/TEX Hit Rate [%]	97.03	Max Bandwidth [%]	58.87
L2 Hit Rate [%]	25.76	Mem Pipes Busy [%]	57.73

Scheduler Statistics

Summary of the activity of the schedulers issuing instructions. Each scheduler maintains a pool of warps that it can issue instructions for. The upper bound of warps in the pool (Theoretical Warps) is limited by the launch configuration. On every cycle each scheduler checks the state of the allocated warps in the pool (Active Warps). Active warps that are not stalled (Eligible Warps) are ready to issue their next instruction. From the set of eligible warps the scheduler selects a single warp from which to issue one or more instructions (Issued Warp). On cycles with no eligible warps, the issue slot is skipped and no instruction is issued. Having many skipped issue slots indicates poor latency hiding.

Active Warps Per Scheduler [warp]	13.54	No Eligible [%]	25.52
Eligible Warps Per Scheduler [warp]	4.31	One or More Eligible [%]	74.48
Issued Warp Per Scheduler	0.74		

Warp State Statistics

Analysis of the states in which all warps spent cycles during the kernel execution. The warp states describe a warp's readiness or inability to issue its next instruction. The warp cycles per instruction define the latency between two consecutive instructions. The higher the value, the more warp parallelism is required to hide this latency. For each warp state, the chart shows the average number of cycles spent in that state per issued instruction. Stalls are not always impacting the overall performance nor are they completely avoidable. Only focus on stall reasons if the schedulers fail to issue every cycle. When executing a kernel with mixed library and user code, these metrics show the combined values.

Warp Cycles Per Issued Instruction [cycle]	18.18	Avg. Active Threads Per Warp	22.90
Warp Cycles Per Executed Instruction [cycle]	18.18	Avg. Not Predicated Off Threads Per Warp	20.93

Instruction Statistics

Statistics of the executed low-level assembly instructions (SASS). The instruction mix provides insight into the types and frequency of the executed instructions. A narrow mix of instruction types implies a dependency on few instruction pipelines, while others remain unused. Using multiple pipelines allows hiding latencies and enables parallel execution. Note that 'Instructions/Opcode' and 'Executed Instructions' are measured differently and can diverge if cycles are spent in system calls.

Executed Instructions [inst]	18,074,880,000	Avg. Executed Instructions Per Scheduler [inst]	56,484,000
Issued Instructions [inst]	18,076,129,251	Avg. Issued Instructions Per Scheduler [inst]	56,487,903.91

Launch Statistics

Summary of the configuration used to launch the kernel. The launch configuration defines the size of the kernel grid, the division of the grid into blocks, and the GPU resources needed to execute the kernel. Choosing an efficient launch configuration maximizes device utilization.

Grid Size	1,440,000	Registers Per Thread [register/thread]	32
Block Size	256	Static Shared Memory Per Block [byte/block]	0
Threads [thread]	368,640,000	Dynamic Shared Memory Per Block [byte/block]	0
Waves Per SM	2,250	Driver Shared Memory Per Block [byte/block]	0
		Shared Memory Configuration Size [byte]	0

Occupancy

Occupancy is the ratio of the number of active warps per multiprocessor to the maximum number of possible active warps. Another way to view occupancy is the percentage of the hardware's ability to process warps that is actively in use. Higher occupancy does not always result in higher performance, however, low occupancy always reduces the ability to hide latencies, resulting in overall performance degradation. Large discrepancies between the theoretical and the achieved occupancy during execution typically indicates highly imbalanced workloads.

Theoretical Occupancy [%]	100	Block Limit Registers [block]	8
Theoretical Active Warps per SM [warp]	64	Block Limit Shared Mem [block]	32
Achieved Occupancy [%]	84.69	Block Limit Warps [block]	8
Achieved Active Warps Per SM [warp]	54.20	Block Limit SM [block]	32

Source Counters

All

Source metrics, including warp stall reasons. Sampling Data metrics are periodically sampled over the kernel runtime. They indicate when warps were stalled and couldn't be scheduled. See the documentation for a description of all stall reasons. Only focus on stalls if the schedulers fail to issue every cycle.