Name: *Tianqi Yang*
NetID: *tianqiy4*
Section: *AL2*

ECE 408/CS483 Milestone 2 Report

1. Show output of rai running Mini-DNN on the basic GPU convolution implementation for batch size of 1k images. This can either be a screen capture or a text copy of the running output. Please do not show the build output. (The running output should be everything including and after the line "*Loading fashion-mnist data...Done*").

Loading fashion-mnist data...Done
Loading model...Done
Conv-GPU==
Layer Time: 71.2756 ms
Op Time: 4.47135 ms
Conv-GPU==
Layer Time: 69.6683 ms
Op Time: 17.8585 ms

Test Accuracy: 0.886

2. For the basic GPU implementation, list Op Times, whole program execution time, and accuracy for batch size of 100, 1k, and 10k images.

Batch Size	Op Time 1	Op Time 2	Total Execution Time	Accuracy
100	*0.410446*	*1.11607*	*15.087956*	*0.86*
1000	*4.47135*	*17.8585*	*163.27375*	*0.886*
10000	*67.4101*	*175.525*	*1531.1931*	*0.8714*

3. List all the kernels that collectively consumed more than 90% of the kernel time and what percentage of the kernel time each kernel did consume (start with the kernel that consumed the most time, then list the next kernel, until you reach 90% or more).

```
CUDA Kernel Statistics (nanoseconds)

Time(%)      Total Time   Instances         Average         Minimum         Maximum  Name
-------  --------------  ----------  --------------  --------------  --------------  ----------------------------------------------------
  100.0       277378983           2     138689491.5        75910239       201468744  conv_forward_kernel
    0.0            2720           2          1360.0            1344            1376  prefn_marker_kernel
    0.0            2592           2          1296.0            1280            1312  do_not_remove_this_kernel
```

4. List all the CUDA API calls that collectively consumed more than 90% of the API time and what percentage of the API time each call did consume (start with the API call that consumed the most time, then list the next call, until you reach 90% or more).

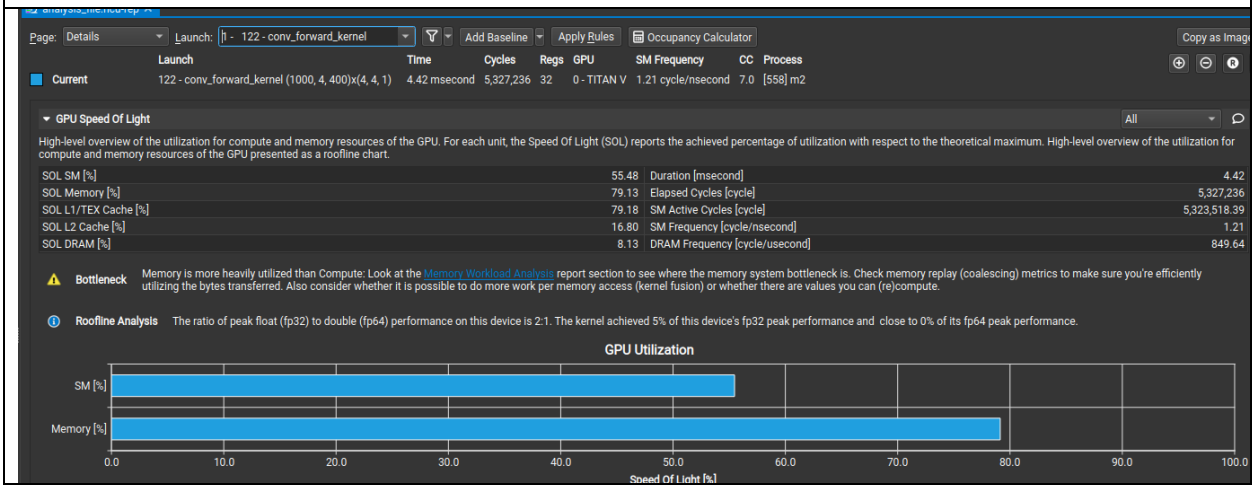
```
CUDA API Statistics (nanoseconds)

Time(%)      Total Time       Calls         Average         Minimum         Maximum  Name
-------  --------------  ----------  --------------  --------------  --------------  ----------------------------------------------------
   65.6      1056489405           8     132061175.6           14104       578524754  cudaMemcpy
   17.2       277402604           6      46233767.3            3019       201471447  cudaDeviceSynchronize
   16.1       259440064           8      32430008.0           99219       247649994  cudaMalloc
```

5. Explain the difference between kernels and CUDA API calls. Please give an example in your explanation for both.

Kernels are the functions we defined usually by ourselves, like conv_forward_kernal.
API calls are the cuda build-in functions that we used, like cudaMemcpy, cudaMalloc

6. Show a screenshot of the GPU SOL utilization




Page: Details
Launch: 4 - 143 - conv_forward_kernel
Add Baseline
Apply Rules
Occupancy Calculator
Copy as Imag
Launch Time Cycles Regs GPU SM Frequency CC Process
Current 143 - conv_forward_kernel (1000, 16, 81)x(4, 4, 1) 17.85 msecond 21,544,913 32 0 - TITAN V 1.21 cycle/nsecond 7.0 [558] 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 [%] 41.34
SOL Memory [%] 48.87
SOL L1/TEX Cache [%] 48.86
SOL L2 Cache [%] 11.08
SOL DRAM [%] 31.12
Duration [msecond] 17.85
Elapsed Cycles [cycle] 21,544,913
SM Active Cycles [cycle] 21,547,740.75
SM Frequency [cycle/nsecond] 1.21
DRAM Frequency [cycle/usecond] 850.78
Bottleneck
This kernel exhibits low compute throughput and memory bandwidth utilization relative to the peak performance of this device. Achieved compute throughput and/or memory bandwidth below 60.0% of peak typically indicate latency issues. Look at Scheduler Statistics and Warp State Statistics for potential reasons.
Roofline Analysis
The ratio of peak float (fp32) to double (fp64) performance on this device is 2:1. The kernel achieved 3% of this device's fp32 peak performance and close to 0% of its fp64 peak performance.
GPU Utilization
SM [%]
Memory [%]
0.0
10.0
20.0
30.0
40.0
50.0
60.0
70.0
80.0
90.0
Speed Of Light [%]