Name: *Haozhe Chen*
NetID: *haozhe3*
Section: *ZJ1*

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 model...Done
Conv-GPU==
Layer Time: 95.2482 ms
Op Time: 4.65613 ms
Conv-GPU==
Layer Time: 96.4938 ms
Op Time: 29.7385 ms

Test Accuracy: 0.886


real    0m9.719s
user    0m9.309s
sys     0m0.372s
```

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

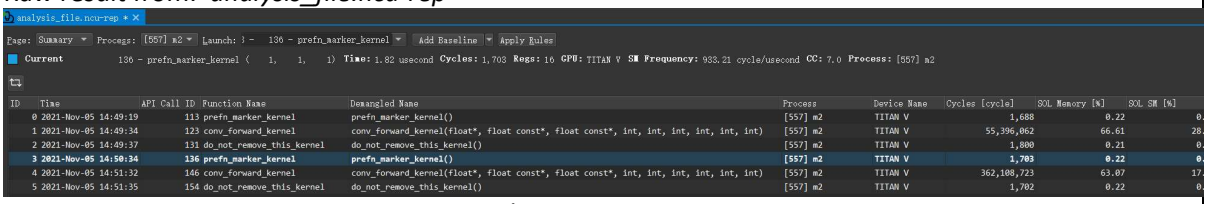
Batch Size	Op Time 1 (ms)	Op Time 2 (ms)	Total Execution Time (s)	Accuracy (%)
100	*0.479951*	*2.93208*	*1.169*	*0.86*
1000	*4.65613*	*29.7385*	*9.719*	*0.886*
10000	*46.1701*	*300.715*	*1m38.061s*	*0.8714*

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).

Ordered result:

ID	Time	API Call ID	Function Name	Demangled Name	Process	Device Name	Cycles [cycle]	SOL Memory [%]	SOL SM [%]	time consume [%]
4	2021-Nov-05 14:51:32	146	conv_forward_kernel	conv_forward_kernel(float*, float const*, float const*, int, int, int, int, int, int)	[557] m2	TITAN V	362,108,723	63.07	17.61	86.7302023
1	2021-Nov-05 14:49:34	123	conv_forward_kernel	conv_forward_kernel(float*, float const*, float const*, int, int, int, int, int, int)	[557] m2	TITAN V	55,396,062	66.61	28.71	13.2681467
2	2021-Nov-05 14:49:37	131	do_not_remove_this_kernel	do_not_remove_this_kernel()	[557] m2	TITAN V	1,800	0.21	0	0.0004311
3	2021-Nov-05 14:50:34	136	prefn_marker_kernel	prefn_marker_kernel()	[557] m2	TITAN V	1,703	0.22	0	0.0004078
5	2021-Nov-05 14:51:35	154	do_not_remove_this_kernel	do_not_remove_this_kernel()	[557] m2	TITAN V	1,702	0.22	0	0.0004076
0	2021-Nov-05 14:49:19	113	prefn_marker_kernel	prefn_marker_kernel()	[557] m2	TITAN V	1,688	0.22	0	0.000404

Raw result from: *analysis_file.ncu-rep*



Raw result from: "nsys profile --stats=true ./m2"

```
Generating CUDA Kernel Statistics...

Generating CUDA Memory Operation Statistics...
CUDA Kernel Statistics (nanoseconds)

Time(%)      Total Time   Instances         Average         Minimum         Maximum  Name
-------  --------------  ----------  --------------  --------------  --------------  ----------------------------------------
  100.0       346749630           2     173374815.0        46157838       300591792  conv_forward_kernel
    0.0            2784           2          1392.0            1376            1408  do_not_remove_this_kernel
    0.0            2688           2          1344.0            1312            1376  prefn_marker_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).

Raw result from: "nsys profile --stats=true ./m2"

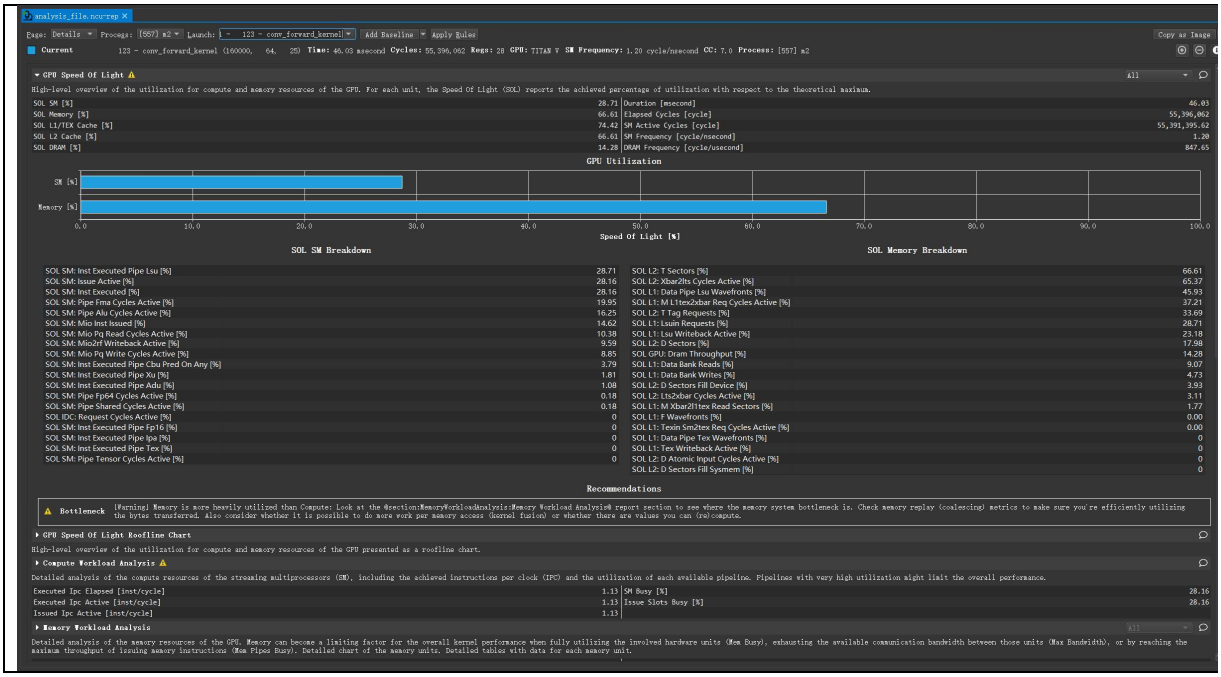
```
Generating CUDA API Statistics...
CUDA API Statistics (nanoseconds)

Time(%)      Total Time     Calls         Average         Minimum         Maximum  Name
-------  --------------  ---------  --------------  --------------  --------------  ----------------------
   67.0      1677332566         10     167733256.6           41465       638221178  cudaMemcpy
   18.3       459089089          8      57386136.1          101787       453266018  cudaMalloc
   13.9       346780285          8      43347535.6             993       300594602  cudaDeviceSynchronize
    0.7        16915467          6       2819244.5           23689        16748407  cudaLaunchKernel
    0.1         3002065          8        375258.1           69622          980092  cudaFree
```

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

Kernels are the code that the GPU actually runs to realize certain functionalities. They will various significantly when the goals of projects change. For example, the function "conv_forward_kerne()l" used in m2.
CUDA API contains some functions that help to connect CPU (or your computer) and GPU. They do some basic things which make it possible for you to control the behavior of GPU, like allocate, free memory, copy data from one to another and Synchronize between devices. For example, the function "cudaDeviceSynchronize()" used in m2.

6. Show a screenshot of the GPU SOL utilization



Output of "nsys profile --stats=true ./m2"

```
Loading model...Done
Conv-GPU==
Layer Time: 1007.22 ms
Op Time: 46.1862 ms
Conv-GPU==
Layer Time: 1012.79 ms
Op Time: 300.628 ms

Test Accuracy: 0.8714

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

        **** WARNING: The collection generated 658146 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 657842 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
-------  ---------------  ---------  --------------  --------------  --------------  ----------------------------
   67.0       1677332566         10     167733256.6           41465       638221178  cudaMemcpy
   18.3        459089089          8      57386136.1          101787       453266018  cudaMalloc
   13.9        346780285          8      43347535.6             993       300594602  cudaDeviceSynchronize
    0.7         16915467          6       2819244.5           23689        16748407  cudaLaunchKernel
    0.1          3002065          8        375258.1           69622          980092  cudaFree



Generating CUDA Kernel Statistics...

Generating CUDA Memory Operation Statistics...
CUDA Kernel Statistics (nanoseconds)

Time(%)      Total Time   Instances         Average         Minimum         Maximum  Name
-------  ---------------  ---------  --------------  --------------  --------------  ----------------------------
  100.0        346749630          2     173374815.0        46157838       300591792  conv_forward_kernel
    0.0             2784          2          1392.0            1376            1408  do_not_remove_this_kernel
    0.0             2688          2          1344.0            1312            1376  prefn_marker_kernel


CUDA Memory Operation Statistics (nanoseconds)

Time(%)      Total Time  Operations         Average         Minimum         Maximum  Name
-------  ---------------  ----------  --------------  --------------  --------------  ----------------------------
   67.4        1120586563           2     560293281.5       483320494       637266069  [CUDA memcpy DtoH]
   32.6         542066049           8      67758256.1            1216       269151198  [CUDA memcpy HtoD]


CUDA Memory Operation Statistics (KiB)

            Total      Operations              Average              Minimum              Maximum  Name
-----------------  --------------  -------------------  -------------------  -------------------  ----------------------------
        2261419.0               8             282677.0                0.004            1000000.0  [CUDA memcpy HtoD]
        1722500.0               2             861250.0              722500.000         1000000.0  [CUDA memcpy DtoH]
```

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

Time(%)      Total Time       Calls         Average         Minimum         Maximum  Name
-------  ---------------  ---------  --------------  --------------  --------------  ----------------------------
   33.3     94027562572         954     98561386.3           49708       100193755  sem_timedwait
   33.3     93894246699         952     98628410.4           75709       100259249  poll
   21.6     60906123006           2   3045306150 3.0     22198308270     38707814736  pthread_cond_wait
   11.7     33009721948          66    500147302.2          500090916       500176847  pthread_cond_timedwait
    0.0       115986185         912       127177.8            1078        21461719  ioctl
    0.0        37690341        9072         4154.6            1179         8683539  read
    0.0        20057132          26       771428.2            1487        19951598  fopen
    0.0         3006839          98        30682.0            1427         1193676  mmap
    0.0         1294766         101        12819.5            7174           48971  open64
    0.0          444303           2       222151.5           36594          407709  pthread_mutex_lock
    0.0          424552           5        84910.4           53027          139217  pthread_create
    0.0          187169           3        62389.7           55785           68150  fgets
    0.0          185216           3        61738.7            8459          155851  fopen64
    0.0          117639          17         6919.9            1293           21175  munmap
    0.0           70904          15         4726.9            2288            7458  write
    0.0           55125           9         6125.0            1032           11298  fflush
    0.0           42396           5         8479.2            6001           11059  open
    0.0           42275          20         2113.8            1078            7284  fclose
    0.0           19699           8         2462.4            1045           10664  fcntl
    0.0           18939           2         9469.5            7580           11359  socket
    0.0           13015           3         4338.3            1137           10454  fwrite
    0.0           10689           1        10689.0           10689           10689  connect
    0.0            7226           1         7226.0            7226            7226  pipe2
    0.0            5371           1         5371.0            5371            5371  pthread_cond_signal
    0.0            1998           1         1998.0            1998            1998  bind
    0.0            1433           1         1433.0            1433            1433  listen



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