Name: *Xing Shen*
NetID: *Xings2*
Section: *ZJU*

ECE 408/CS483 Milestone 3 Report

0. List Op Times, whole program execution time, and accuracy for batch size of 100, 1k, and 5k images from your basic forward convolution kernel in milestone 2. This will act as your baseline this milestone. Note: **Do not** use batch size of 10k when you profile in --*queue rai_amd64_exclusive*. We have limited resources, so any tasks longer than 3 minutes will be killed. Your baseline M2 implementation should comfortably finish in 3 minutes with a batch size of 5k (About 1m35 seconds, with nv-nsight).

Batch Size	Op Time 1	Op Time 2	Total Execution Time	Accuracy
100	*0.193803ms*	*0.897425ms*	*2.353s*	*0.86*
1000	*1.54005ms*	*7.83892ms*	*11.314s*	*0.886*
5000	*8.45031ms*	*43.2912ms*	*1m0.117s*	*0.871*

1. **Optimization 1:** ***Tiled shared memory convolution (2 points)***

   a. Which optimization did you choose to implement and why did you choose that optimization technique.

      This optimization can utilize GPU shared memory efficiently, and in that case the running time should be reduced.

   b. How does the optimization work? Did you think the optimization would increase performance of the forward convolution? Why? Does the optimization synergize with any of your previous optimizations?

      The first optimization, and I believe that this can increase the performance of the forward convolution. It will not synergize with previous optimization.

c. List the Op Times, whole program execution time, and accuracy for batch size of 100, 1k, and 5k images using this optimization (including any previous optimizations also used).

Batch Size	Op Time 1	Op Time 2	Total Execution Time	Accuracy
100	*0.213878ms*	*0.843126ms*	*1.794s*	*0.86*
1000	*2.02368ms*	*8.31878ms*	*11.293s*	*0.886*
5000	*10.0167ms*	*41.0016ms*	*55.566s*	*0.871*

d. Was implementing this optimization successful in improving performance? Why or why not? Include profiling results from *nsys* and *Nsight-Compute* to justify your answer, directly comparing to your baseline (or the previous optimization this one is built off of).

We could found that the runtime of conv_forward_kernel is smaller than m2, which means shared memory will increase the performancy.

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

Time(%)      Total Time       Calls         Average         Minimum         Maximum  Name
-------  --------------  ----------  --------------  --------------  --------------  --------------------------------------------
   98.3       186824376          12      15568698.0            3020       172067514  cudaMalloc
    1.1         2179042          12        181586.8           47681          431282  cudaMemcpy
    0.4          747596          12         62299.7            4690          150348  cudaFree
    0.1          244589           4         61147.2           19748          106851  cudaDeviceSynchronize
    0.1          115178           4         28794.5           26079           32047  cudaLaunchKernel



Generating CUDA Kernel Statistics...

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

Time(%)      Total Time   Instances         Average         Minimum         Maximum  Name
-------  --------------  ----------  --------------  --------------  --------------  --------------------------------------------
  100.0           83328           4         20832.0           13312           26144  conv_forward_kernel


CUDA Memory Operation Statistics (nanoseconds)

Time(%)      Total Time  Operations         Average         Minimum         Maximum  Name
-------  --------------  ----------  --------------  --------------  --------------  --------------------------------------------
   85.7          693661           8         86707.6            1151          282975  [CUDA memcpy HtoD]
   14.3          115775           4         28943.8           13088           46432  [CUDA memcpy DtoH]


CUDA Memory Operation Statistics (KiB)

            Total      Operations            Average            Minimum            Maximum  Name
-----------------  --------------  -----------------  -----------------  -----------------  ----------------------------
           6740.0               8              842.0              0.316             2677.0  [CUDA memcpy HtoD]
           1422.0               4              355.0            148.535              577.0  [CUDA memcpy DtoH]
```

e. What references did you use when implementing this technique?

Lecture, textbook.

2. **Optimization 2: *Weight matrix in constant memory (0.5 points)***

a. Which optimization did you choose to implement and why did you choose that optimization technique.

Weight matrix in constant memory
The matrix in CNN is fixed, so use constant memory will easily increase the performance.

b. How does the optimization work? Did you think the optimization would increase performance of the forward convolution? Why? Does the optimization synergize with any of your previous optimizations?

   I believe it will increase the performance because we just change the memory type into constant memory. This type of memory could be read more quickly, leading to a shorter time.

c. List the Op Times, whole program execution time, and accuracy for batch size of 100, 1k, and 5k images using this optimization (including any previous optimizations also used).

Batch Size	Op Time 1	Op Time 2	Total Execution Time	Accuracy
100	*0.173929ms*	*0.660685ms*	*1.686s*	*0.86*
1000	*1.77945ms*	*7.13333ms*	*11.830s*	*0.886*
5000	*8.79106ms*	*32.3771ms*	*54.501s*	*0.871*

d. Was implementing this optimization successful in improving performance? Why or why not? Include profiling results from *nsys* and *Nsight-Compute* to justify your answer, directly comparing to your baseline (or the previous optimization this one is built off of).

Constant memory can be accessed faster than global memory, so the increase of performance is obvious.

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

Time(%)      Total Time       Calls         Average         Minimum         Maximum  Name
-------  --------------  ----------  --------------  --------------  --------------  ----------------------------------------
   70.6       586512328          14      41893737.7           27040       311492168  cudaMemcpy
   21.4       177736789          14      12695484.9            2679       174458196  cudaMalloc
    5.4        44626435          10       4462643.5            3389        35679965  cudaDeviceSynchronize
    2.2        18108053          10       1810805.3           24868        17863175  cudaLaunchKernel
    0.3         2580126          20        129006.3            1442          467076  cudaFree
    0.1          981294           6        163549.0          106150          197473  cudaMemcpyToSymbol




Generating CUDA Kernel Statistics...

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

Time(%)      Total Time   Instances         Average         Minimum         Maximum  Name
-------  --------------  ----------  --------------  --------------  --------------  ----------------------------------------
  100.0        44598865           6       7433144.2           11296        35670337  conv_forward_kernel
    0.0            2976           2          1488.0            1408            1568  do_not_remove_this_kernel
    0.0            2592           2          1296.0            1248            1344  prefn_marker_kernel


CUDA Memory Operation Statistics (nanoseconds)

Time(%)      Total Time  Operations         Average         Minimum         Maximum  Name
-------  --------------  ----------  --------------  --------------  --------------  ----------------------------------------
   92.1       535212687           6      89202114.5           23231       310676930  [CUDA memcpy DtoH]
    7.9        45918090          14       3279863.6            1152        24001760  [CUDA memcpy HtoD]


CUDA Memory Operation Statistics (KiB)

            Total      Operations            Average            Minimum            Maximum  Name
-----------------  --------------  -----------------  -----------------  -----------------  ------------------------------
         862672.0               6           143778.7            148.535           500000.0  [CUDA memcpy DtoH]
         276206.0              14            19729.0              0.004           144453.0  [CUDA memcpy HtoD]
```

e. What references did you use when implementing this technique?

   Textbook, Lecture.

3. **Optimization 3:** ***Using Streams to overlap computation with data transfer (4 points)***

   a. Which optimization did you choose to implement and why did you choose that optimization technique.

      Stream is a part without appearing in MPs, so I try to use this optimization in order to cement my knowledge.

   b. How does the optimization work? Did you think the optimization would increase performance of the forward convolution? Why? Does the optimization synergize with any of your previous optimizations?

Using Streams, we can overlap the data transfer, which means different part of data could be transferred at the same time, which obviously increase the performance. This optimization is based on ms2, since we change the arrangement of functions, and we delete two functions, which means OP time cannot be used to judge the performance. Also, there is no synergize with another optimization.

c. List the Op Times, whole program execution time, and accuracy for batch size of 100, 1k, and 5k images using this optimization (including any previous optimizations also used).

Batch Size	Op Time 1	Op Time 2	Total Execution Time	Accuracy
100	/	/	/	/
1000	/	/	/	/
5000	/	/	/	/

d. Was implementing this optimization successful in improving performance? Why or why not? Include profiling results from *nsys* and *Nsight-Compute* to justify your answer, directly comparing to your baseline (or the previous optimization this one is built off of).

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

Time(%)      Total Time       Calls         Average         Minimum         Maximum  Name
-------  --------------  ----------  --------------  --------------  --------------  ------------------------------------------------
   68.2       554062126          20      27703106.3           29765       287236369  cudaMemcpy
   23.0       187155552          20       9357777.6            2804       182727273  cudaMalloc
    6.4        51868289          10       5186828.9            3784        43354229  cudaDeviceSynchronize
    2.1        16895553          10       1689555.3           25874        16629417  cudaLaunchKernel
    0.4         2870725          20        143536.2            3212          440050  cudaFree



Generating CUDA Kernel Statistics...

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

Time(%)      Total Time   Instances         Average         Minimum         Maximum  Name
-------  --------------  ----------  --------------  --------------  --------------  ------------------------------------------------
  100.0        51838542           6       8639757.0            8704        43347285  conv_forward_kernel
    0.0            2752           2          1376.0            1376            1376  do_not_remove_this_kernel
    0.0            2560           2          1280.0            1216            1344  prefn_marker_kernel


CUDA Memory Operation Statistics (nanoseconds)

Time(%)      Total Time  Operations         Average         Minimum         Maximum  Name
-------  --------------  ----------  --------------  --------------  --------------  ------------------------------------------------
   92.3       500467907           6      83411317.8           12703       286504962  [CUDA memcpy DtoH]
    7.7        41959040          14       2997074.3            1120        20732530  [CUDA memcpy HtoD]


CUDA Memory Operation Statistics (KiB)

            Total      Operations            Average            Minimum            Maximum  Name
-----------------  --------------  -----------------  -----------------  -----------------  ------------------------------------------------
         862672.0               6           143778.7            148.535           500000.0  [CUDA memcpy DtoH]
         276206.0              14            19729.0              0.004           144453.0  [CUDA memcpy HtoD]
```

Since the stream is small, and if the stream is larger, there will be error with the code. In that case, the performance actually is almost the same as the ms2.

e. What references did you use when implementing this technique?

Textbook, lecture.

4. **Optimization 4: Input channel reduction: atomics (2 points)**

   a. Which optimization did you choose to implement and why did you choose that optimization technique.

   I choose this optimization because it could synergize with the pervious optimization perfectively.

   b. How does the optimization work? Did you think the optimization would increase performance of the forward convolution? Why? Does the optimization synergize with any of your previous optimizations?

   Using atomic reduction, each output element on feature map, a thread will be used to calculate the convolution values.
   This optimization can synergize with previous optimizations.

   c. List the Op Times, whole program execution time, and accuracy for batch size of 100, 1k, and 5k images using this optimization (including any previous optimizations also used).

Batch Size	Op Time 1	Op Time 2	Total Execution Time	Accuracy
100	*0.185732ms*	*0.707704ms*	*1.799s*	*0.86*
1000	*1.79141ms*	*7.16269ms*	*10.891s*	*0.886*
5000	*8.83665ms*	*35.6465ms*	*53.014s*	*0.871*

   d. Was implementing this optimization successful in improving performance? Why or why not? Include profiling results from *nsys* and *Nsight-Compute* to justify your answer, directly comparing to your baseline (or the previous optimization this one is built off of).

From the total execution time, we could easily find that using atomicAdd is much faster than not use it.

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

Time(%)      Total Time       Calls         Average         Minimum         Maximum  Name
-------  --------------  ----------  --------------  --------------  --------------  --------------------
   69.6       572649949          14      40903567.8           30453       284302584  cudaMemcpy
   22.4       184178091          14      13155577.9            2993       181031417  cudaMalloc
    5.4        44748093          10       4474809.3            3501        35721326  cudaDeviceSynchronize
    2.1        17272196          10       1727219.6           25019        17023355  cudaLaunchKernel
    0.3         2728018          20        136400.9            1435          466551  cudaFree
    0.1          987088           6        164514.7          105460          184772  cudaMemcpyToSymbol



Generating CUDA Kernel Statistics...

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

Time(%)      Total Time   Instances         Average         Minimum         Maximum  Name
-------  --------------  ----------  --------------  --------------  --------------  --------------------
  100.0        44718941           6       7453156.8           11488        35717482  conv_forward_kernel
    0.0            2784           2          1392.0            1376            1408  do_not_remove_this_kernel
    0.0            2688           2          1344.0            1344            1344  prefn_marker_kernel


CUDA Memory Operation Statistics (nanoseconds)

Time(%)      Total Time  Operations         Average         Minimum         Maximum  Name
-------  --------------  ----------  --------------  --------------  --------------  --------------------
   91.7       520008102           6      86668017.0           23232       283589437  [CUDA memcpy DtoH]
    8.3        47032111          14       3359436.5            1152        24005926  [CUDA memcpy HtoD]


CUDA Memory Operation Statistics (KiB)

            Total      Operations            Average            Minimum            Maximum  Name
-----------------  --------------  -----------------  -----------------  -----------------  --------------------
         862672.0               6           143778.7            148.535           500000.0  [CUDA memcpy DtoH]
         276206.0              14            19729.0              0.004           144453.0  [CUDA memcpy HtoD]
```

Also from the nsys, we could find the time of conv_forward_kernel is shorter than the original.

e. What references did you use when implementing this technique?

   Textbook, Lecture.

5. **Optimization 5: *FP16 arithmetic (4 points)***

   a. Which optimization did you choose to implement and why did you choose that optimization technique.

      FP16 arithmetic
      Since this will reduce the size of data, which may cause reduction on data transfer.

   b. How does the optimization work? Did you think the optimization would increase performance of the forward convolution? Why? Does the optimization synergize with any of your previous optimizations?

Use FP16 will change the data type from float to half, which will reduce the data transfer time and calculate time. In that case, the performance of total code will increase.
Besides, this optimization will also slightly increase the accuracy.

c. List the Op Times, whole program execution time, and accuracy for batch size of 100, 1k, and 5k images using this optimization (including any previous optimizations also used).

Batch Size	Op Time 1	Op Time 2	Total Execution Time	Accuracy
100	*0.564685ms*	*1.90058ms*	*1.611s*	*0.86*
1000	*7.18244ms*	*19.0528ms*	*12.059s*	*0.887*
5000	*18.1903ms*	*90.2399ms*	*53.424s*	*0.8712*

d. Was implementing this optimization successful in improving performance? Why or why not? Include profiling results from *nsys* and *Nsight-Compute* to justify your answer, directly comparing to your baseline (or the previous optimization this one is built off of).

Actually it improves accuracy slight, and it will short the total execution at specific batch size.
I use this optimization based on ms2, so we compare it with

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

Time(%)      Total Time       Calls         Average         Minimum         Maximum  Name
-------  --------------  ----------  --------------  --------------  --------------  --------------------
   61.6       530649924          20      26532496.2           29416       280246210  cudaMemcpy
   23.2       199834194          38       5258794.6            2361       194887229  cudaMalloc
   12.6       108791118          34       3199738.8             815        88857354  cudaDeviceSynchronize
    1.9        16011356          28        571834.1            4521        15631290  cudaLaunchKernel
    0.7         6119828          38        161048.1            2588         2239751  cudaFree




Generating CUDA Kernel Statistics...

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

Time(%)      Total Time   Instances         Average         Minimum         Maximum  Name
-------  --------------  ----------  --------------  --------------  --------------  --------------------
   94.8       103011767           6      17168627.8            8640        88846938  conv_forward_kernel
    4.0         4351756           6        725292.7            1920         2531188  half2float
    1.2         1341178          12        111764.8            1568          702141  float2half
    0.0            2816           2          1408.0            1408            1408  do_not_remove_this_kernel
    0.0            2624           2          1312.0            1248            1376  prefn_marker_kernel


CUDA Memory Operation Statistics (nanoseconds)

Time(%)      Total Time  Operations         Average         Minimum         Maximum  Name
-------  --------------  ----------  --------------  --------------  --------------  --------------------
   91.2       478569827           6      79761637.8           23552       279546645  [CUDA memcpy DtoH]
    8.8        45912453          14       3279460.9            1120        24009774  [CUDA memcpy HtoD]


CUDA Memory Operation Statistics (KiB)

            Total      Operations            Average            Minimum            Maximum  Name
-----------------  --------------  -----------------  -----------------  -----------------  --------------------
         862672.0               6           143778.7            148.535           500000.0  [CUDA memcpy DtoH]
         276206.0              14            19729.0              0.004           144453.0  [CUDA memcpy HtoD]
```

e. What references did you use when implementing this technique?

Textbook, lectures.

Total points: 2 + 0.5 + 4 + 2 + 4 = 12.5

The code is in the folder optimize.