Name: *Yize Chen*
NetID: *yizec2*
Section: *AL2*

ECE 408/CS483 Milestone 3 Report

0. List Op Times, whole program execution time, and accuracy for batch size of 100, 1k, and 10k images from your basic forward convolution kernel in milestone 2. This will act as your baseline this milestone.

Batch Size	Op Time 1	Op Time 2	Total Execution Time	Accuracy
100	*0.199 ms*	*0.832864 ms*	*0m1.178s*	*0.86*
1000	*1.85528 ms*	*8.20502 ms*	*0m10.028s*	*0.886*
10000	*18.3851 ms*	*79.0952 ms*	*1m37.096s*	*0.8714*

1. **Optimization 1:** ***Tiled shared memory convolution***

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

    Tiled shared memory convolution.
    With tiled shared memory convolution we can save the bandwidth of accessing global memory.

    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 separate the output images into tiles with size TILE_WIDTH * TILE_WIDTH, and each block calculates one TILE with each thread calculates one pixel.*
Yes.
The bandwidth of accessing global memory is saved.
N/A.

c. List the Op Times, whole program execution time, and accuracy for batch size of 100, 1k, and 10k images using this optimization (including any previous optimizations also used).

Batch Size	Op Time 1	Op Time 2	Total Execution Time	Accuracy
100	*0.175439ms*	*0.709415 ms*	*0m1.136s*	*0.86*
1000	*1.66172 ms*	*7.1029 ms*	*0m9.519s*	*0.886*
10000	*16.3433 ms*	*70.9712 ms*	*1m33.436s*	*0.8714*

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).

Yes. The op time decreases by around 12.5%, and the total time also decreases.

Time(%)	Total Time	Instances	Average	Minimum	Maximum	Name
100.0	100304688	2	50152344.0	18391606	81913082	conv_forward_kernel

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.

SOL SM [%]	0.00	Duration [usecond]	1.82
SOL Memory [%]	0.23	Elapsed Cycles [cycle]	1,743
SOL L1/TEX Cache [%]	80.27	SM Active Cycles [cycle]	3.74
SOL L2 Cache [%]	0.23	SM Frequency [cycle/usecond]	954.95
SOL DRAM [%]	0	DRAM Frequency [cycle/usecond]	666.67

(Base nsys and n-compute)

Time(%)	Total Time	Instances	Average	Minimum	Maximum	Name
100.0	87384451	2	43692225.5	16411481	70972970	conv_forward_kernel

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.

SOL SM [%]	0.00	Duration [usecond]	1.79
SOL Memory [%]	0.23	Elapsed Cycles [cycle]	1,671
SOL L1/TEX Cache [%]	94.86	SM Active Cycles [cycle]	3.16
SOL L2 Cache [%]	0.23	SM Frequency [cycle/usecond]	931.92
SOL DRAM [%]	0	DRAM Frequency [cycle/usecond]	666.67

(Current nsys and n-compute)
The time of kernel function reduced by around 12.5%.
The duration time reduced as well.

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

3rd-Edition-Chapter16-case-study-DNN-FINAL-corrected.pdf from ECE 408 course website

2. **Optimization 2: *Shared memory matrix multiplication and input matrix unrolling***

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

Shared memory matrix multiplication and input matrix unrolling. Because each input tile won't have to be loaded M times so that global memory bandwidth is saved.

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?

Unroll the input batch of images first such that we can use shared memory matrix multiplication between the kernel and the unrolled matrix to calculate output.
Yes. Because each input tile won't have to be loaded M times so that global memory bandwidth is saved.
No.

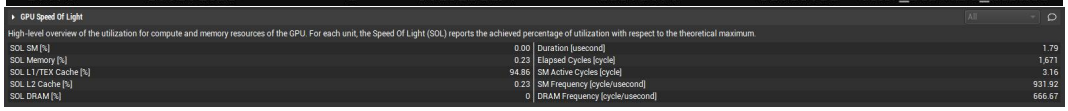
c. List the Op Times, whole program execution time, and accuracy for batch size of 100, 1k, and 10k images using this optimization (including any previous optimizations also used).

Batch Size	Op Time 1	Op Time 2	Total Execution Time	Accuracy
100	*1.44911 ms*	*1.92679 ms*	*0m1.080s*	*0.86*
1000	*22.7752 ms*	*17.2486 ms*	*0m9.126s*	*0.886*
10000	*209.508 ms*	*160.534 ms*	*1m31.951s*	*0.8714*

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).

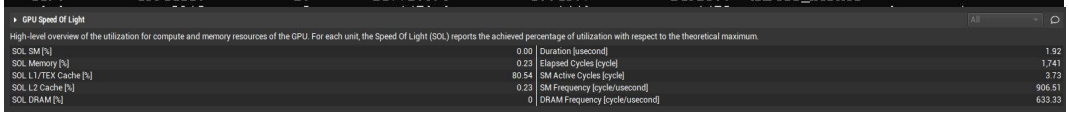
No, the op times are larger.

Time(%)	Total Time	Instances	Average	Minimum	Maximum	Name
100.0	100304688	2	50152344.0	18391606	81913082	conv_forward_kernel



(Base nsys and n-compute)

Time(%)	Total Time	Instances	Average	Minimum	Maximum	Name
88.1	323457304	20	16172865.2	14000189	18395454	conv_forward_mult_mat_unroll_kernel
11.9	43544110	20	2177205.5	1904307	2452399	unroll_kernel



(current)

Kernel time and duration time are larger.

It could because of the space needed is too large so that I have to use 10 loops to divide the 10k-batch into 10 1-k batch. And launching 2 kernels takes more time.

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

3rd-Edition-Chapter16-case-study-DNN-FINAL-corrected.pdf from ECE 408 course website

3. **Optimization 3:** ***Kernel fusion for unrolling and matrix-multiplication***
(Delete this section blank if you did not implement this many optimizations.)

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

Kernel fusion for unrolling and matrix-multiplication.
Because the previous optimization method performs badly, with fused kernel, we don't have to launch two kernels and we don't need to read from and write to a new global memory space for unrolled matrix.

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?

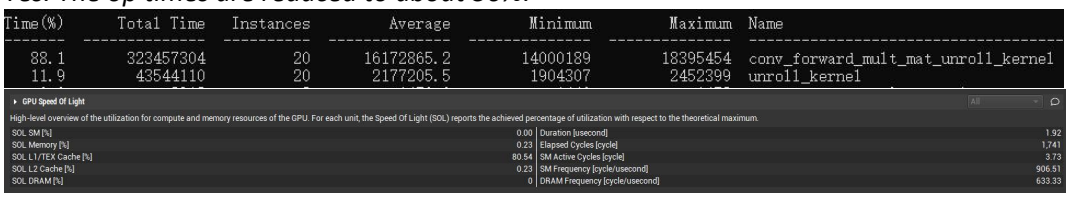
I fused the unroll kernel and the matrix multiplication kernel into one kernel.
Yes. Because we don't have to launch two kernels and we don't need to read from and write to a new global memory space for unrolled matrix.
Yes, it synergizes with (2).

c. List the Op Times, whole program execution time, and accuracy for batch size of 100, 1k, and 10k images using this optimization (including any previous optimizations also used).

Batch Size	Op Time 1	Op Time 2	Total Execution Time	Accuracy
100	*0.928001 ms*	*0.610537 ms*	*0m1.141s*	*0.86*
1000	*9.13618 ms*	*6.60243 ms*	*0m9.995s*	*0.886*
10000	*94.3528 ms*	*65.6543 ms*	*1m35.990s*	*0.8714*

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).

Yes. The op times are reduced to about 50%.



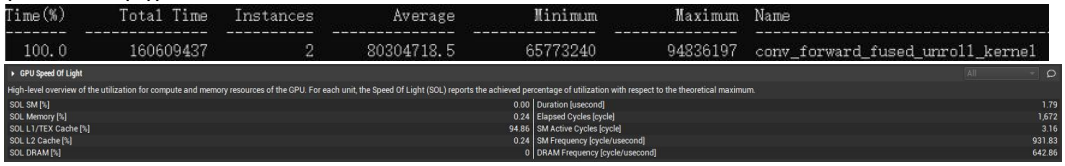
Time(%)	Total Time	Instances	Average	Minimum	Maximum	Name
88.1	323457304	20	16172865.2	14000189	18395454	conv_forward_mult_mat_unroll_kernel
11.9	43544110	20	2177205.5	1904307	2452399	unroll_kernel

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.

SOL SM [%]	0.00	Duration [usecond]	1.92
SOL Memory [%]	0.23	Elapsed Cycles [cycle]	1,741
SOL L1/TEX Cache [%]	80.54	SM Active Cycles [cycle]	3.73
SOL L2 Cache [%]	0.23	SM Frequency [cycle/usecond]	906.51
SOL DRAM [%]	0	DRAM Frequency [cycle/usecond]	633.33

(Previous (2))



Time(%)	Total Time	Instances	Average	Minimum	Maximum	Name
100.0	160609437	2	80304718.5	65773240	94836197	conv_forward_fused_unroll_kernel

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.

SOL SM [%]	0.00	Duration [usecond]	1.79
SOL Memory [%]	0.24	Elapsed Cycles [cycle]	1,672
SOL L1/TEX Cache [%]	94.86	SM Active Cycles [cycle]	3.16
SOL L2 Cache [%]	0.24	SM Frequency [cycle/usecond]	931.83
SOL DRAM [%]	0	DRAM Frequency [cycle/usecond]	642.86

The kernel time is reduced to about 50% and duration time decreases.

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

None.

4. **Optimization 4:** ***Weight matrix in constant memory***
   (Delete this section blank if you did not implement this many optimizations.)

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

      Weight matrix in constant memory. Because I think the weight matrix is read by every thread and I can easily save shared memory and the time to copy data from global memory to shared memory

   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 constant cache for weight matrix.
      Yes. Because the weight matrix is read by every thread. Loading it to constant memory saves shared memory space and the time to copy data from global memory to shared memory.
      Yes, Tiled shared memory convolution(1).

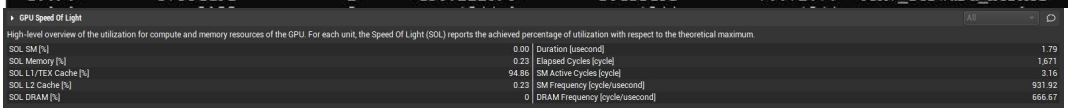
   c. List the Op Times, whole program execution time, and accuracy for batch size of 100, 1k, and 10k images using this optimization (including any previous optimizations also used).

Batch Size	Op Time 1	Op Time 2	Total Execution Time	Accuracy
100	*0.165567 ms*	*0.651862 ms*	*0m1.084s*	*0.86*
1000	*1.48668 ms*	*6.4176 ms*	*0m9.116s*	*0.886*
10000	*14.5567 ms*	*64.1886 ms*	*1m29.439s*	*0.8714*

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).

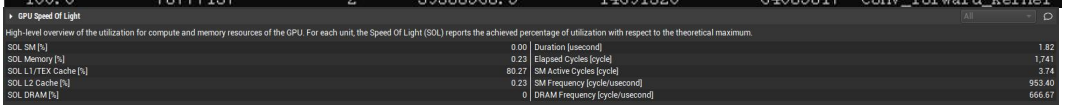
Yes.
The op times are reduced by around 9%.

Time(%)	Total Time	Instances	Average	Minimum	Maximum	Name
100.0	87384451	2	43692225.5	16411481	70972970	conv_forward_kernel



(Previous (1))

Time(%)	Total Time	Instances	Average	Minimum	Maximum	Name
100.0	78777137	2	39388568.5	14691320	64085817	conv_forward_kernel



The kernel time reduced by around 10%.

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

Course slides -- ECE 408: Lecture 7.

5. **Optimization 5:** ***Using Streams to overlap computation with data transfer***
(Delete this section if you did not implement this many optimizations.)

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

Using Streams to overlap computation with data transfer. Because using multiple streams could save the time for memory copy and reduce the global memory size.

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 2 streams in turn that the second stream copies, does kernel and copies back overlaps right after the first stream.
Yes. Because using multiple streams could save the time for memory copy and reduce the global memory size.
Yes. 2 and 4.

c. List the Op Times, whole program execution time, and accuracy for batch size of 100, 1k, and 10k images using this optimization (including any previous optimizations also used).

Batch Size	Op Time 1	Op Time 2	Total Execution Time	Accuracy
100	*6.08013 ms*	*4.87303 ms*	*0m1.087s*	*0.86*
1000	*57.8538 ms*	*45.1731 ms*	*0m9.121s*	*0.886*
10000	*590.739 ms*	*435.935 ms*	*1m29.785s*	*0.8714*

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).

No. The performance is very similar to previous method. And note the reason op times are too large is because of the way to measure it. It is actually very close to layer time. But the total time is normal and even reduced a little.
The reason the performance didn't increase a lot could be that the 500-batch I use for one stream is too small that parallelism is not fully achieved.

Time(%)	Total Time	Instances	Average	Minimum	Maximum	Name
100.0	78777137	2	39388568.5	14691320	64085817	conv_forward_kernel

▸ 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.

SOL SM [%]	0.00	Duration [usecond]	1.82
SOL Memory [%]	0.23	Elapsed Cycles [cycle]	1,741
SOL L1/TEX Cache [%]	80.27	SM Active Cycles [cycle]	3.74
SOL L2 Cache [%]	0.23	SM Frequency [cycle/usecond]	953.40
SOL DRAM [%]	0	DRAM Frequency [cycle/usecond]	666.67

(Previous (4))

Time(%)	Total Time	Instances	Average	Minimum	Maximum	Name
100.0	79145663	40	1978641.6	740541	3231410	conv_forward_kernel

▸ 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.

SOL SM [%]	0.00	Duration [usecond]	1.79
SOL Memory [%]	0.24	Elapsed Cycles [cycle]	1,670
SOL L1/TEX Cache [%]	94.49	SM Active Cycles [cycle]	3.17
SOL L2 Cache [%]	0.24	SM Frequency [cycle/usecond]	930.80
SOL DRAM [%]	0	DRAM Frequency [cycle/usecond]	654.76

(Current).
The time of kernel is a little larger that previous method.

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

Lecture 22 slides.
Introduction of creation of streams from
https://developer.nvidia.com/zh-cn/blog/how-overlap-data-transfers-cuda-cc/