Name: *Ryan Wolf*
NetID: *ryantw2*
Section: *AL1*

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
100	*0.985946 ms*	*2.10156 ms*	*0m2.803s*
1000	*1.60082 ms*	*6.14408*	*0m10.360s*
10000	*15.7936 ms*	*61.1303 ms*	*1m39.915s*

1 **Optimization 1: A*n advanced matrix multiplication algorithm (register-tiled, for example) (5 points)***
2.a Which optimization did you choose to implement and why did you choose that optimization technique.
I did register tiling matrix multiplication with in place unrolling. I did this technique because I had heard from previous lectures that this was the fastest matrix multiplication technique and I wanted to improve the data reuse in the convolution.
2.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?

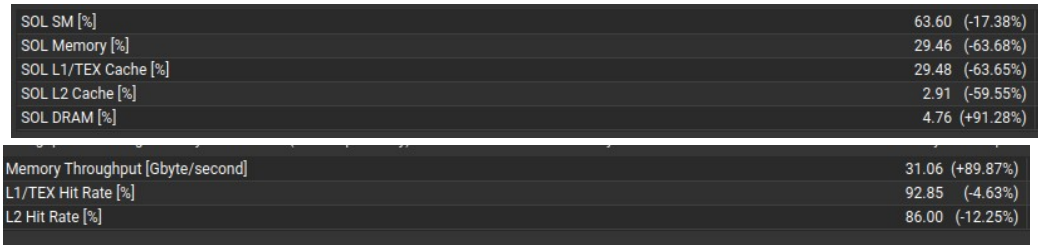
The optimization (as I have implemented it) works as follows. Tiles of the mask "matrix" are loaded into shared memory, and each thread is responsible for computing a column of data in the output "matrix." The threads go row by row loading entries of the image "matrix' into registers and computing with the tiles in shared memory. I thought the optimization would improve performance of the forward convolution since it allows for better data reuse than the default algorithm. I have no previous optimizations so there is no synergy.

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
100	*0.459465 ms*	*0.348025 ms*	*0m1.237s*
1000	*4.3582 ms*	*3.22706 ms*	*0m10.008s*
10000	*41.237 ms*	*31.6985 ms*	*1m38.287s*

2.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).

Performance was improved for the second call of the kernel because of the larger mask size. In the second kernel, the DRAM SOL and memory throughput increase by roughly 90% over the baseline as shown below:



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

I used slides from ECE 508 to figure out how register tiling worked, along with a reference implementation from C. Pearson's lectures.

2 **Optimization 2: *Fixed point (FP16) arithmetic. (note this can modify model accuracy slightly) (4 points)***

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

I chose the FP16 implementation because I wanted to improve the computation speed of the kernel.

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 optimization works by combining two 32 bit floats into a single __half2 datatype and performed the multiply and add part of the matrix on the new datatype. The conversion happens on the fly as tiles are loaded. I think the optimization will increase performance because now the kernel will be able to do twice as many floating point multiplications and additions compared to before. It is not related to my previous optimization, but I imagine that it will improve it nonetheless.

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
100	*0.339943 ms*	*0.276251 ms*	*0m1.164s*
1000	*3.19052 ms*	*2.33471 ms*	*0m10.068s*
10000	*31.5057 ms*	*22.7146 ms*	*1m38.568s*

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, performance was improved over both the baseline and the previous optimization. If I compare SOL performance, the new optimization increases dramatically over the old optimization in many areas as shown below. This improvement happens in both the first and second kernel calls.

Comparing first kernel calls:

SOL SM [%]	74.11 (+14.21%)
SOL Memory [%]	25.04 (-10.86%)
SOL L1/TEX Cache [%]	25.04 (-10.79%)
SOL L2 Cache [%]	5.65 (+36.65%)
SOL DRAM [%]	6.28 (+35.98%)

Memory Throughput [Gbyte/second]	40.98 (+35.79%)
L1/TEX Hit Rate [%]	85.52 (+1.42%)
L2 Hit Rate [%]	92.02 (+0.26%)

Comparing second kernel calls:

SOL SM [%]	72.67 (+14.26%)
SOL Memory [%]	29.88 (+1.42%)
SOL L1/TEX Cache [%]	29.71 (+0.78%)
SOL L2 Cache [%]	4.19 (+44.30%)
SOL DRAM [%]	6.60 (+38.69%)
Memory Throughput [Gbyte/second]	43.07 (+38.69%)
L1/TEX Hit Rate [%]	94.07 (+1.31%)
L2 Hit Rate [%]	86.58 (+0.68%)

e What references did you use when implementing this technique?

I referenced the cuda math api for the arithmetic and data conversion functions.

3 **Optimization 3: *Multiple kernel implementations for different layer sizes (1 point)***

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

I chose to implement different kernel implementations for different layer sizes. The key difference is in the number of rows of the mask matrix that are loaded at a time. I noticed that the mask matrix used in the first kernel only has 4 rows, while my code tries to tile it in chunks of 16 rows. So, I made a separate kernel that does only 4 rows at a time.

3.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 optimization checks how many rows will be in the mask matrix, and if there are fewer than 16 rows it uses the kernel that loads fewer rows at a time. I believe it will increase performance because now there will be less warp divergence from threads trying to load items that don't exist in the matrix.

3.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
100	*0.294512 ms*	*0.257333 ms*	*0m1.144s*
1000	*2.46596 ms*	*2.31691 ms*	*0m10.153s*
10000	*24.3856 ms*	*22.722 ms*	*1m44.921s*

3.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, performance was greatly improved. I will only focus on the first kernel since that was the only thing to change from the last optimization. The SOL performance increased noticeably, but more importantly the scheduler vastly improved as shown below.

SOL SM [%]	79.39 (+7.11%)
SOL Memory [%]	13.77 (-45.01%)
SOL L1/TEX Cache [%]	13.77 (-45.00%)
SOL L2 Cache [%]	6.18 (+9.28%)
SOL DRAM [%]	7.46 (+18.73%)

Active Warps Per Scheduler [warp]	11.82 (+74.47%)
Eligible Warps Per Scheduler [warp]	4.28 (+103.35%)
Issued Warp Per Scheduler	0.79 (+6.72%)

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

I only referenced my previous implementation of the kernel.

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

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

I chose to implement streams because I knew that it would inherently benefit the performance by overlapping the matrix multiplication with the data transfer.

4.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 optimization works by having 3 streams transferring data to and from the GPU while also performing computation on data that is on the GPU. The batch of images is split up into chunks that are copied over one by one.

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
100	*0.003429 ms*	*0.002858 ms*	*0m1.139s*
1000	*0.004843 ms*	*0.005105 ms*	*0m10.361s*
10000	*0.005065 ms*	*0.004973 ms*	*1m41.106s*

4.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).

It was successful, but it's hard to see from the nsight performance. All metrics are lower, but the layer time is much faster.

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

I referenced the slides on streams from the lecture.