Name: *Ding-Hsiang Huang*
NetID: *dhhuang3*
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	*0.225599ms*	*0.9883 ms*	*1.213899ms*	*0.86*
1000	*2.20112ms*	*9.03543ms*	*11.23655ms*	*0.886*
10000	*21.6349ms*	*87.2559 ms*	*108.8908ms*	*0.8714*

1. **Optimization 1: Sweeping various parameters to find best values**

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

    Change BLOCK_WIDTH from 32 to 16.
    And it doesn't need to modify code too much.

    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?

    Changing value of parameters and then compare and choose the best result.
    I think it increase performance of the forward convolution, because BLOCK_WIDTH would make affect to kernel very much.
    It didn't synergize with any of my previous optimizations

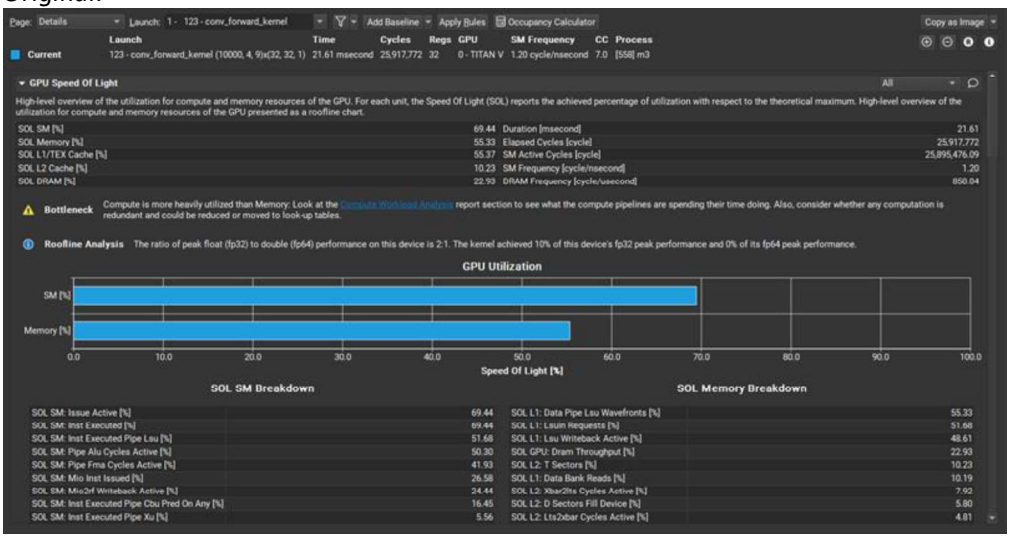
c. List the Op Times, whole program execution time, and accuracy for batch size of 100, 1k, and 10k images using this optimization (including any previous optimizations also used).

Batch Size	Op Time 1	Op Time 2	Total Execution Time	Accuracy
100	*0.176167ms*	*0.637506 ms*	*0.83673ms*	*0.86*
1000	*1.65375ms*	*6.32831ms*	*7.98206ms*	*0.886*
10000	*16.1505 ms*	*63.687 ms*	*79.8375ms*	*0.8714*

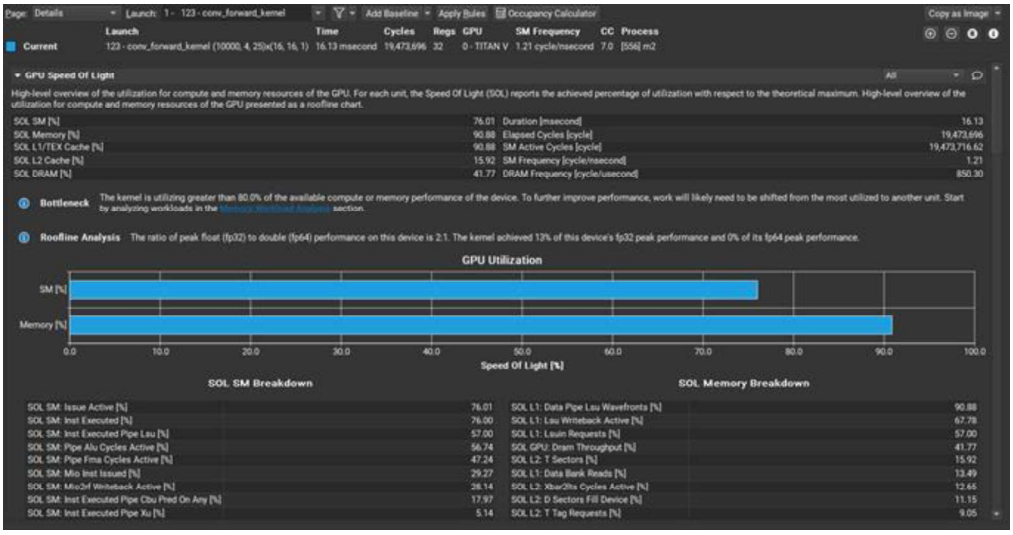
d. Was implementing this optimization successful in improving performance? Why or why not? Include profiling results from *nsys* and *Nsight-Compute* to justify your answer, directly comparing to your baseline (or the previous optimization this one is built off of).

It increase performance by 27%.

Original:



This one:



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

Course lecture slides.

2. **Optimization 2: *Weight matrix(kernel values) in constant memory***

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

Weight matrix(kernel values) in constant memory.
It's because that is efficient and easy to implement.

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 store kernel value in a constant array.

I think it would increase performance.

It didn't synergize with any of my previous optimizations

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).

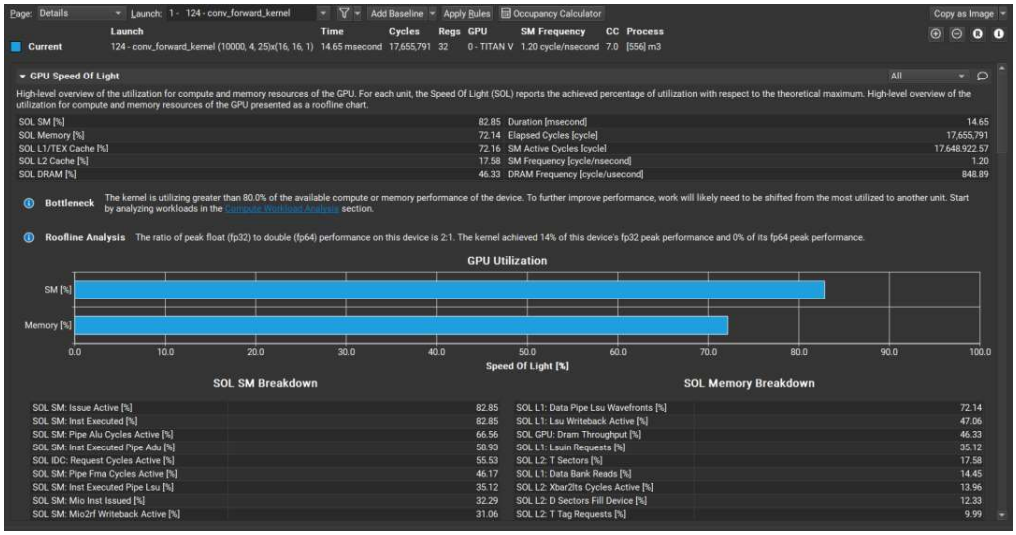
Batch Size	Op Time 1	Op Time 2	Total Execution Time	Accuracy
100	*0.168653ms*	*0.577544ms*	*0.746197ms*	*0.86*
1000	*1.49388ms*	*5.70478ms*	*7.19866*	*0.886*
10000	*13.4229ms*	*51.8432ms*	*65.2661ms*	*0.8714*

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).

It increases its performance by 18%.

Previous:

This one:



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

Reference MP4

3. **Optimization 3: Tuning with restrict and loop unrolling**

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

      Restrict and loop unrolling.
      Implementation is easy, while kernel got more efficient very well.

   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 unroll two bottom for loop in kernel.

      I think it would increase.

      It didn't synergize with any of my previous optimizations

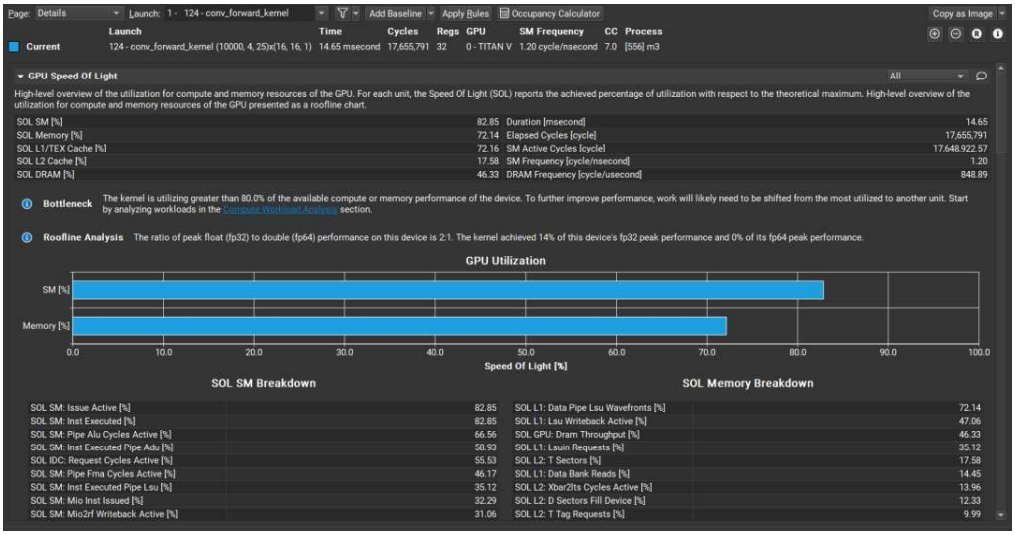
   c. List the Op Times, whole program execution time, and accuracy for batch size of 100, 1k, and 10k images using this optimization (including any previous optimizations also used).

Batch Size	Op Time 1	Op Time 2	Total Execution Time	Accuracy
100	*0.120819ms*	*0.405108ms*	*0.525927ms*	*0.86*
1000	*1.08286ms*	*3.89197ms*	*4.97483*	*0.886*
10000	*10.7273ms*	*37.1066ms*	*47.8339ms*	*0.8714*

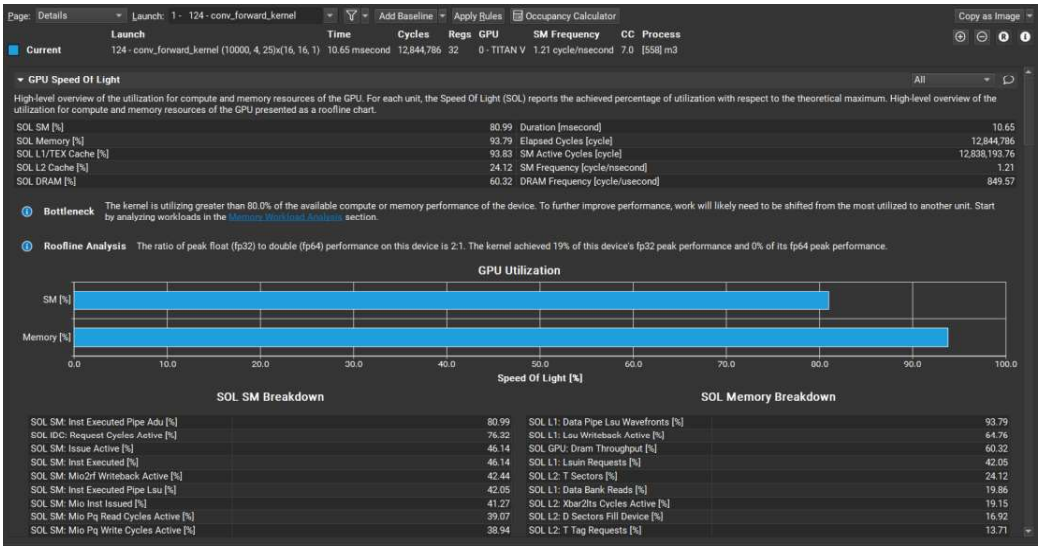
d. Was implementing this optimization successful in improving performance? Why or why not? Include profiling results from *nsys* and *Nsight-Compute* to justify your answer, directly comparing to your baseline (or the previous optimization this one is built off of).

It increases its performance by 27%.

Previous:



This one:



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

https://en.wikipedia.org/wiki/Loop_unrolling

4. **Optimization 4:** ***Using Streams to overlap computation with data transfer***

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

      Using Streams to overlap computation with data transfer.
      It seems alike pipeline, so I assume it would improve the speed.

   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?

      Replace memcpy and modify kernel code with the offset argument.

      I think it would increase.

      It didn't synergize with any of my previous optimizations

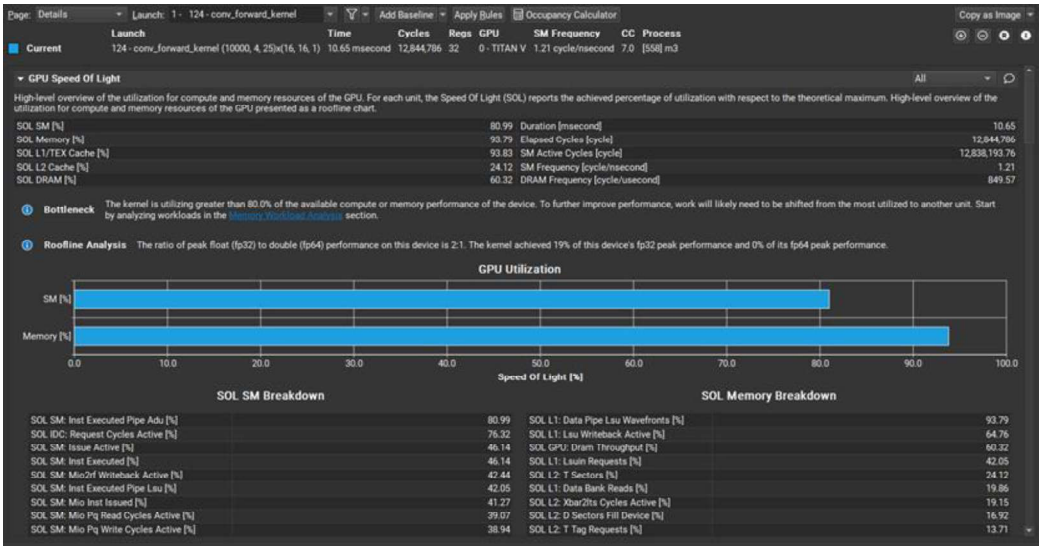
   c. List the Op Times, whole program execution time, and accuracy for batch size of 100, 1k, and 10k images using this optimization (including any previous optimizations also used).

Batch Size	Op Time 1	Op Time 2	Total Execution Time	Accuracy
100	*0.137434ms*	*0.60971ms*	*0.747144ms*	*0.86*
1000	*1.07529ms*	*4.63217ms*	*5.70746ms*	*0.886*
10000	*10.4623ms*	*37.9429ms*	*48.4052ms*	*0.8714*

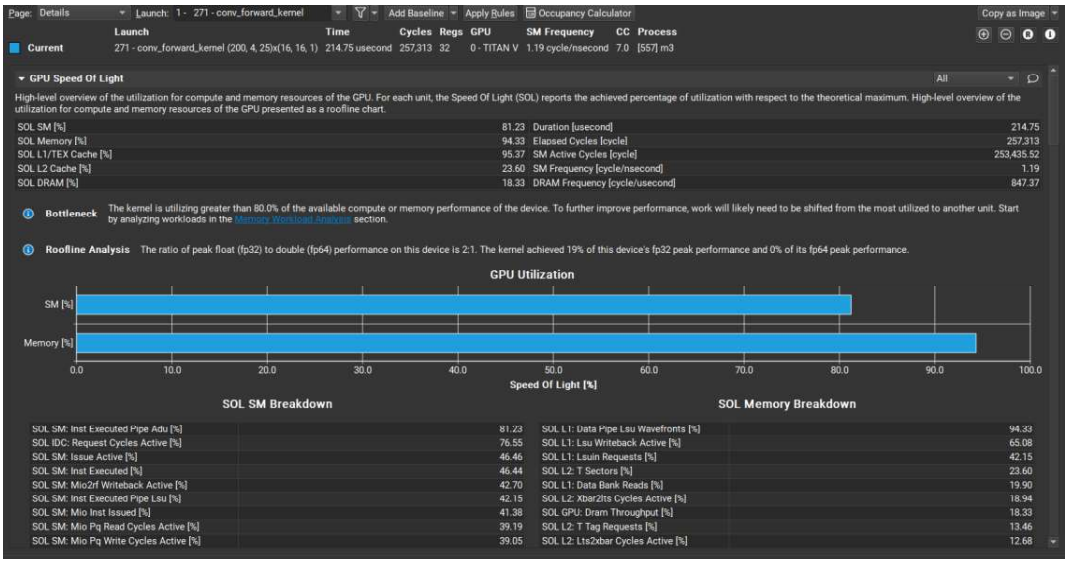
d. Was implementing this optimization successful in improving performance? Why or why not? Include profiling results from *nsys* and *Nsight-Compute* to justify your answer, directly comparing to your baseline (or the previous optimization this one is built off of).

From the total execution time, it doesn't improve performance.

Previous:



This one:



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

Nvidia developer website.
https://developer.nvidia.com/blog/how-overlap-data-transfers-cuda-cc/

5. **Optimization 5:** ***Multiple kernel implementations for different layer sizes***

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

      Multiple kernel implementations for different layer sizes.
      There are two layers with different sizes.

   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 created another kernel and modify both of them.
      I think it would increase performance.

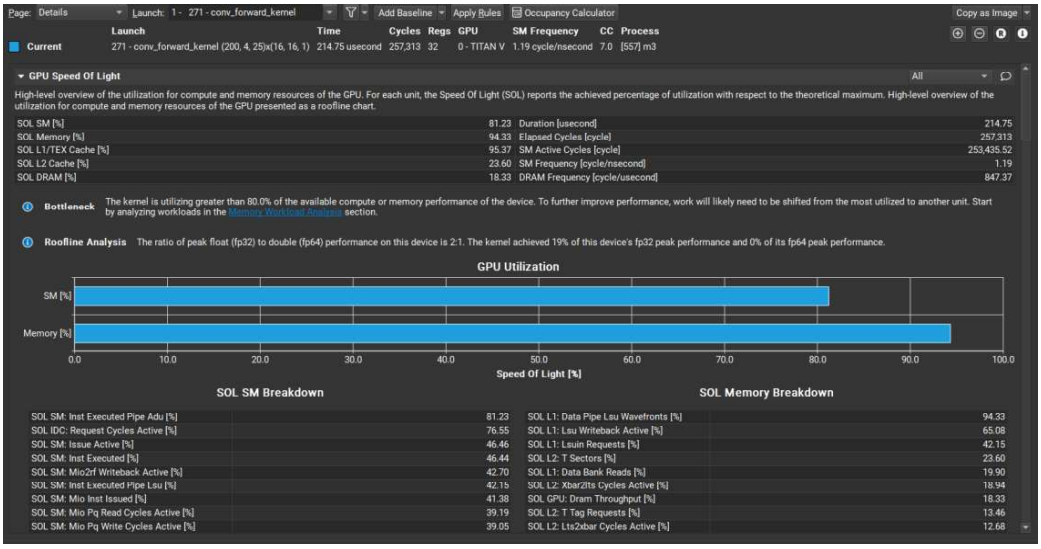
   c. List the Op Times, whole program execution time, and accuracy for batch size of 100, 1k, and 10k images using this optimization (including any previous optimizations also used).

Batch Size	Op Time 1	Op Time 2	Total Execution Time	Accuracy
100	*0.133309ms*	*0.653882ms*	*0.787191ms*	*0.86*
1000	*1.04395ms*	*5.41446ms*	*6.45841ms*	*0.886*
10000	*10.1003ms*	*36.4697ms*	*46.57ms*	*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).

From the total execution time, it improved a little.

Previous:



This one:

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

Course lectures.