**Name:** *Aditya Gupta*
**NetID:** *adityag5*
**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 | Accuracy |
|---|---|---|---|---|
| 100 | *0.32 ms* | *1.197 ms* | *1.169 s* | *0.86* |
| 1000 | *3.08 ms* | *11.985 ms* | *9.941 s* | *0.886* |
| 10000 | *30.44 ms* | *120.418 ms* | *36.117 s* | *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.

   *I chose the shared memory convolution as an optimization technique as it would be the most straightforward to implement starting from the baseline version.*

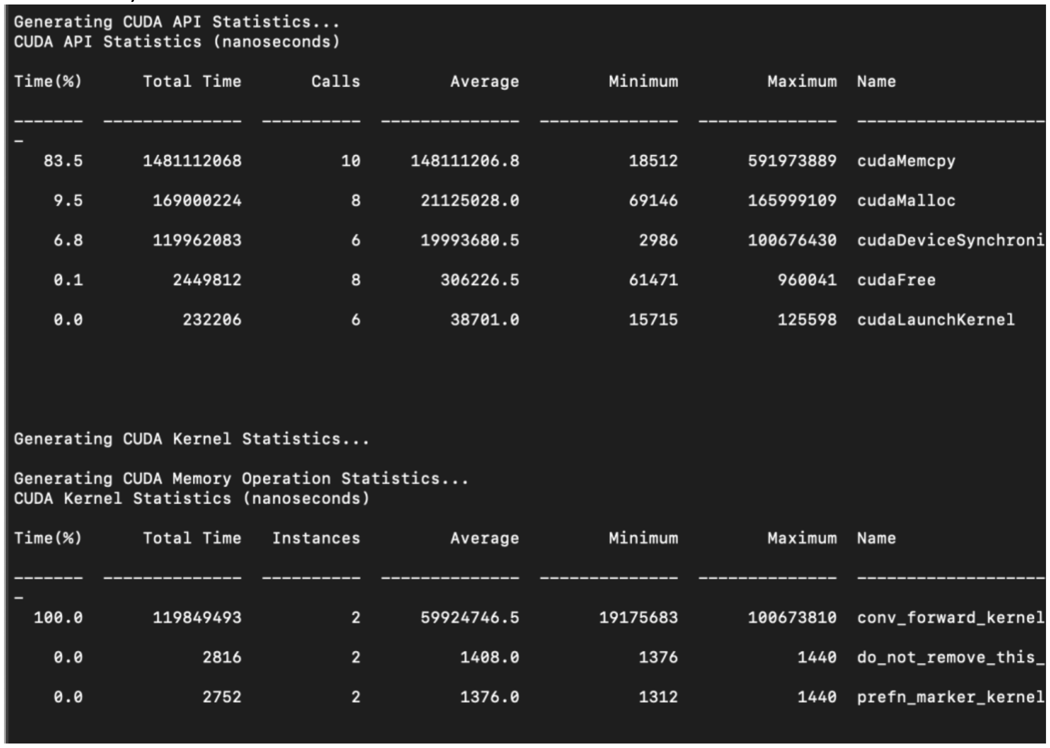
   b. How does the optimization work? Did you think the optimization would increase performance of the forward convolution? Why? Does the optimization synergize with any of your previous optimizations?

*This optimization uses shared memory, which is shared by all threads in a given block, which results in lower latency and higher bandwidth than using global memory, in turn resulting in better performance of the kernel. This technique involves storing both input data and kernel data into shared memory before convolution is done, and since global memory is usually slow, this technique leads to a faster convolution process.*

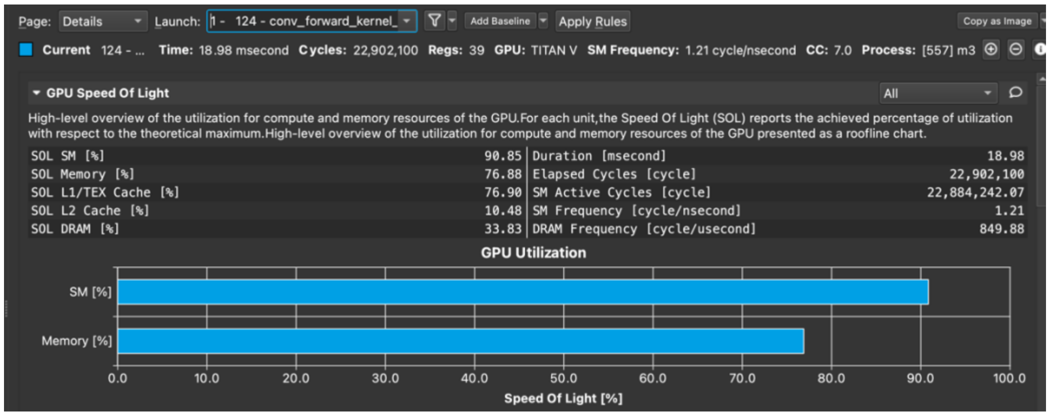
c. List the Op Times, whole program execution time, and accuracy for batch size of 100, 1k, and 10k images using this optimization (including any previous optimizations also used).

| Batch Size | Op Time 1 | Op Time 2 | Total Execution Time | Accuracy |
|---|---|---|---|---|
| 100 | *0.203 ms* | *1.1016 ms* | *1.085 ms* | *0.86* |
| 1000 | *1.93 ms* | *10.075 ms* | *9.224 s* | *0.886* |
| 10000 | *19.002 ms* | *100.42 ms* | *32.989 s* | *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).



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

Time(%)      Total Time       Calls         Average         Minimum         Maximum  Name

-------  --------------  ----------  --------------  --------------  --------------  ------------------
-
   83.5      1481112068          10     148111206.8           18512       591973889  cudaMemcpy

    9.5       169000224           8      21125028.0           69146       165999109  cudaMalloc

    6.8       119962083           6      19993680.5            2986       100676430  cudaDeviceSynchroni

    0.1         2449812           8        306226.5           61471          960041  cudaFree

    0.0          232206           6         38701.0           15715          125598  cudaLaunchKernel




Generating CUDA Kernel Statistics...

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

Time(%)      Total Time   Instances         Average         Minimum         Maximum  Name

-------  --------------  ----------  --------------  --------------  --------------  ------------------
-
  100.0       119849493           2      59924746.5        19175683       100673810  conv_forward_kernel

    0.0            2816           2          1408.0            1376            1440  do_not_remove_this_

    0.0            2752           2          1376.0            1312            1440  prefn_marker_kernel
```

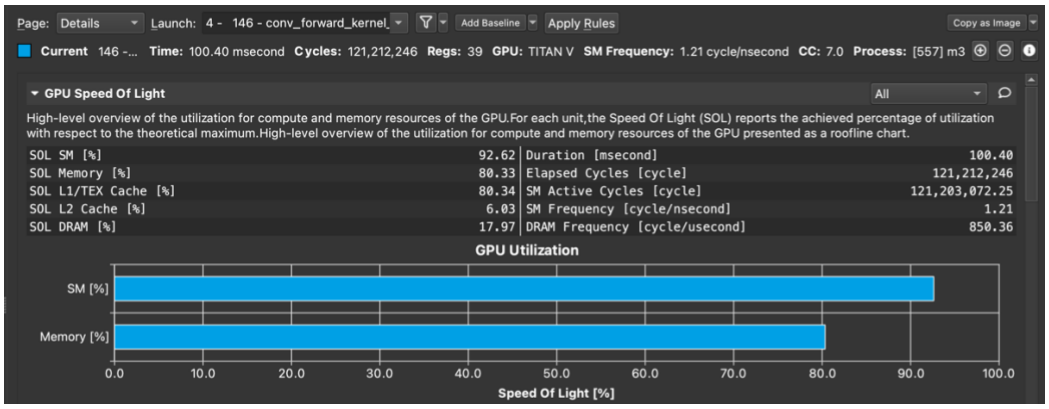

Page: Details  Launch: 1 - 124 - conv_forward_kernel_  Add Baseline  Apply Rules  Copy as Image

Current 124 - ... Time: 18.98 msecond Cycles: 22,902,100 Regs: 39 GPU: TITAN V SM Frequency: 1.21 cycle/nsecond CC: 7.0 Process: [557] m3

GPU Speed Of Light  All

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.High-level overview of the utilization for compute and memory resources of the GPU presented as a roofline chart.

| | | | |
|---|---|---|---|
| SOL SM [%] | 90.85 | Duration [msecond] | 18.98 |
| SOL Memory [%] | 76.88 | Elapsed Cycles [cycle] | 22,902,100 |
| SOL L1/TEX Cache [%] | 76.90 | SM Active Cycles [cycle] | 22,884,242.07 |
| SOL L2 Cache [%] | 10.48 | SM Frequency [cycle/nsecond] | 1.21 |
| SOL DRAM [%] | 33.83 | DRAM Frequency [cycle/usecond] | 849.88 |


Page: Details  Launch: 4 - 146 - conv_forward_kernel_  Add Baseline  Apply Rules  Copy as Image

Current 146 -... Time: 100.40 msecond Cycles: 121,212,246 Regs: 39 GPU: TITAN V SM Frequency: 1.21 cycle/nsecond CC: 7.0 Process: [557] m3

GPU Speed Of Light  All

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.High-level overview of the utilization for compute and memory resources of the GPU presented as a roofline chart.

| | | | |
|---|---|---|---|
| SOL SM [%] | 92.62 | Duration [msecond] | 100.40 |
| SOL Memory [%] | 80.33 | Elapsed Cycles [cycle] | 121,212,246 |
| SOL L1/TEX Cache [%] | 80.34 | SM Active Cycles [cycle] | 121,203,072.25 |
| SOL L2 Cache [%] | 6.03 | SM Frequency [cycle/nsecond] | 1.21 |
| SOL DRAM [%] | 17.97 | DRAM Frequency [cycle/usecond] | 850.36 |

*Yes, this optimization was successful in improving performance as seen from the clear reduction in Op times and execution times. However, it still hasn't met the desired performance of Op times less than 70 ms. The switch from using global memory entirely to using shared memory for partitioning data clearly reduced total execution time, which is the most apparent in the 10,000 dataset.*

*The SOL utilization screenshot also shows how much the SM usage % has increased and the memory usage % has decreased, indicating that performance was improved when using shared memory.*

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

*https://developer.download.nvidia.com/compute/cuda/1.1-Beta/x86_64_website/projects/convolutionSeparable/doc/convolutionSeparable.pdf*

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.

      *I chose loop unrolling as my second technique as to reap the benefits of instruction level benefits at the expense of space complexity.*

   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?

      *This is also a standalone optimization utilizing shared memory matrix multiplication and unrolling of the input matrix. For this optimization, both the input data and kernel data are stored in shared memory, which has the same benefits as optimization 1. On top of that, before convolution, the input data X will be modified as an unrolled matrix to introduce instruction level parallelism which improves performance.*

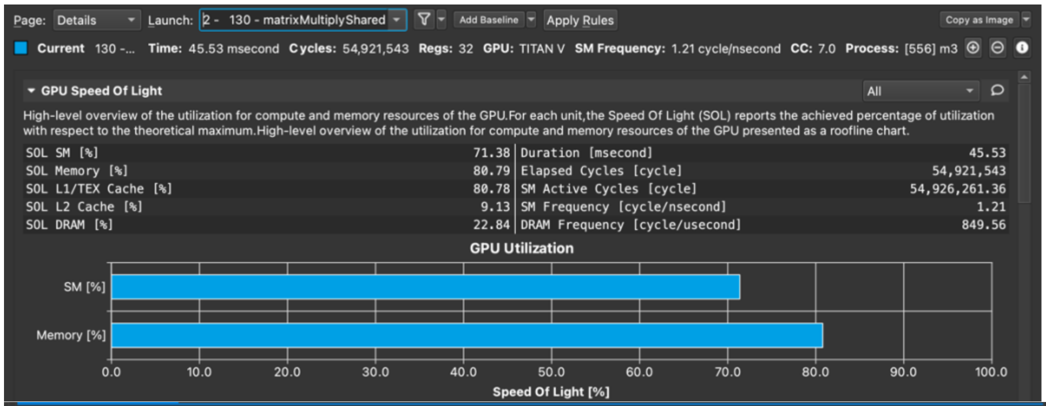
   c. List the Op Times, whole program execution time, and accuracy for batch size of 100, 1k, and 10k images using this optimization (including any previous optimizations also used).

| Batch Size | Op Time 1 | Op Time 2 | Total Execution Time | Accuracy |
|---|---|---|---|---|
| 100 | *1.51 ms* | *1.196 ms* | *1.159 s* | *0.86* |
| 1000 | *13.02 ms* | *9.57 ms* | *10.138 s* | *0.886* |
| 10000 | *130.8 ms* | *92.94 ms* | *38.1 s* | *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).

*This optimization was not successful in improving in improving performance as suggested by the OP times and execution times which have increased from the baseline for all batch sizes.*

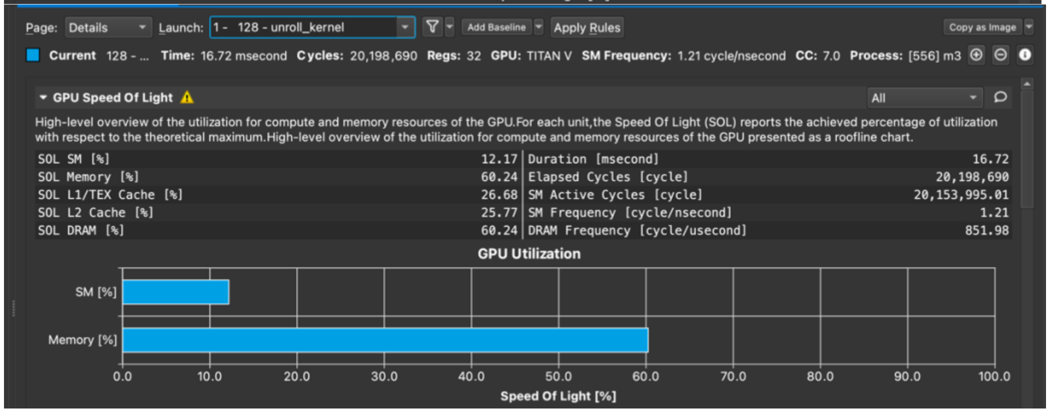
*The SOL utilization screenshots are also indicators of the poor performance as when the duration of the two kernels is added, they are much slower than the baseline duration. This indicates that these two kernels separately don't improve performance much but need to be fused together.*



Current 130 -... Time: 45.53 msecond Cycles: 54,921,543 Regs: 32 GPU: TITAN V SM Frequency: 1.21 cycle/nsecond CC: 7.0 Process: [556] m3

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.High-level overview of the utilization for compute and memory resources of the GPU presented as a roofline chart.

```
SOL SM [%]                 71.38 | Duration [msecond]                  45.53
SOL Memory [%]             80.79 | Elapsed Cycles [cycle]         54,921,543
SOL L1/TEX Cache [%]       80.78 | SM Active Cycles [cycle]    54,926,261.36
SOL L2 Cache [%]            9.13 | SM Frequency [cycle/nsecond]         1.21
SOL DRAM [%]               22.84 | DRAM Frequency [cycle/usecond]     849.56
```



Current 128 - ... Time: 16.72 msecond Cycles: 20,198,690 Regs: 32 GPU: TITAN V SM Frequency: 1.21 cycle/nsecond CC: 7.0 Process: [556] m3

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.High-level overview of the utilization for compute and memory resources of the GPU presented as a roofline chart.

```
SOL SM [%]                 12.17 | Duration [msecond]                  16.72
SOL Memory [%]             60.24 | Elapsed Cycles [cycle]         20,198,690
SOL L1/TEX Cache [%]       26.68 | SM Active Cycles [cycle]    20,153,995.01
SOL L2 Cache [%]           25.77 | SM Frequency [cycle/nsecond]         1.21
SOL DRAM [%]               60.24 | DRAM Frequency [cycle/usecond]     851.98
```

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

*[https://www.nvidia.com/docs/IO/116711/sc11-unrolling-parallel-loops.pdf](https://www.nvidia.com/docs/IO/116711/sc11-unrolling-parallel-loops.pdf)*
*[https://www.seas.upenn.edu/~cis565/Lectures2011S/Lecture12.pdf](https://www.seas.upenn.edu/~cis565/Lectures2011S/Lecture12.pdf)*
*ECE 408 MPs*

## 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.

*Since I already implemented loop unrolling in the previous optimization, this optimization involving restrict was the most sensible to follow up with.*

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?

   *This optimization involves tuning the kernel with the restrict keywork and also reaping the benefits of loop unrolling. Adding the restrict keyword before the pointers to both the input matrices leads to the kernel to utilize the read-only data cache on the GPU. Loop unrolling will be done by adding the #pragma unroll before the for loops for the compiler to unroll the loops. This builds off the previous optimization since loop unrolling is being used.*

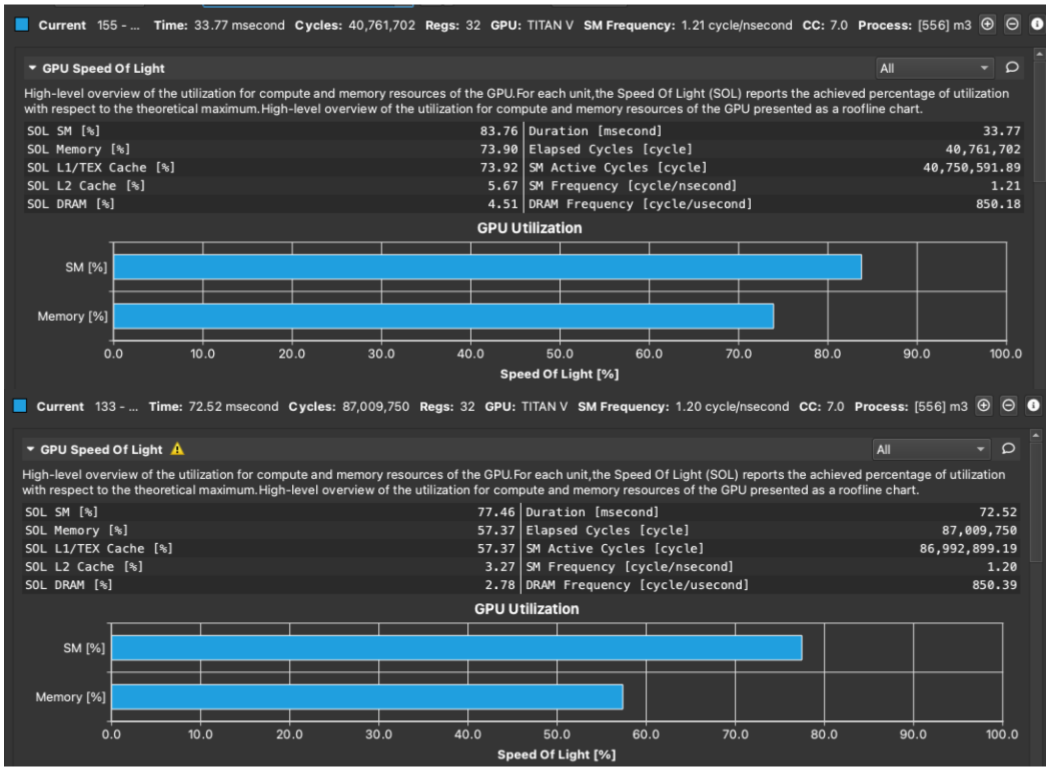
c. List the Op Times, whole program execution time, and accuracy for batch size of 100, 1k, and 10k images using this optimization (including any previous optimizations also used).

| Batch Size | Op Time 1 | Op Time 2 | Total Execution Time | Accuracy |
|---|---|---|---|---|
| 100 | *0.744 ms* | *0.361 ms* | *1.228 s* | *0.86* |
| 1000 | *7.28 ms* | *3.403 ms* | *9..96 s* | *0.886* |
| 10000 | *71.6 ms* | *33.795 ms* | *38.3 s* | *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, this optimization improved performance significantly as suggested from the significant reductions in OP times and execution times in all batch sizes. This suggests that the loop unrolling in the kernel caused parallelism in instruction level tasks and caused to improvement in performance. Moreover, the memory usage was significantly less in both kernel calls as seen by the Memory usage % in the SOL screenshots below indicating that the __restrict__ keyword also helped reduce total memory accesses.*

*Moreover the SM usage % increased from about 40 % to about 80 % indicating better GPU usage resulting in a convolution operation with greater performance.*


**Current** 155 - ... **Time:** 33.77 msecond **Cycles:** 40,761,702 **Regs:** 32 **GPU:** TITAN V **SM Frequency:** 1.21 cycle/nsecond **CC:** 7.0 **Process:** [556] m3

**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.High-level overview of the utilization for compute and memory resources of the GPU presented as a roofline chart.

| | | | |
|---|---|---|---|
| SOL SM [%] | 83.76 | Duration [msecond] | 33.77 |
| SOL Memory [%] | 73.90 | Elapsed Cycles [cycle] | 40,761,702 |
| SOL L1/TEX Cache [%] | 73.92 | SM Active Cycles [cycle] | 40,750,591.89 |
| SOL L2 Cache [%] | 5.67 | SM Frequency [cycle/nsecond] | 1.21 |
| SOL DRAM [%] | 4.51 | DRAM Frequency [cycle/usecond] | 850.18 |

**Current** 133 - ... **Time:** 72.52 msecond **Cycles:** 87,009,750 **Regs:** 32 **GPU:** TITAN V **SM Frequency:** 1.20 cycle/nsecond **CC:** 7.0 **Process:** [556] m3

**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.High-level overview of the utilization for compute and memory resources of the GPU presented as a roofline chart.

| | | | |
|---|---|---|---|
| SOL SM [%] | 77.46 | Duration [msecond] | 72.52 |
| SOL Memory [%] | 57.37 | Elapsed Cycles [cycle] | 87,009,750 |
| SOL L1/TEX Cache [%] | 57.37 | SM Active Cycles [cycle] | 86,992,899.19 |
| SOL L2 Cache [%] | 3.27 | SM Frequency [cycle/nsecond] | 1.20 |
| SOL DRAM [%] | 2.78 | DRAM Frequency [cycle/usecond] | 850.39 |

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

https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#restrict
https://forums.developer.nvidia.com/t/pragma-unroll/3042

4. **Optimization 4: Sweeping parameters to find best values**

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

*From the baseline kernel, this was a pretty simple optimization to implement as it would only involve fine tuning parameters such as block sizes, tile widths.*

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?

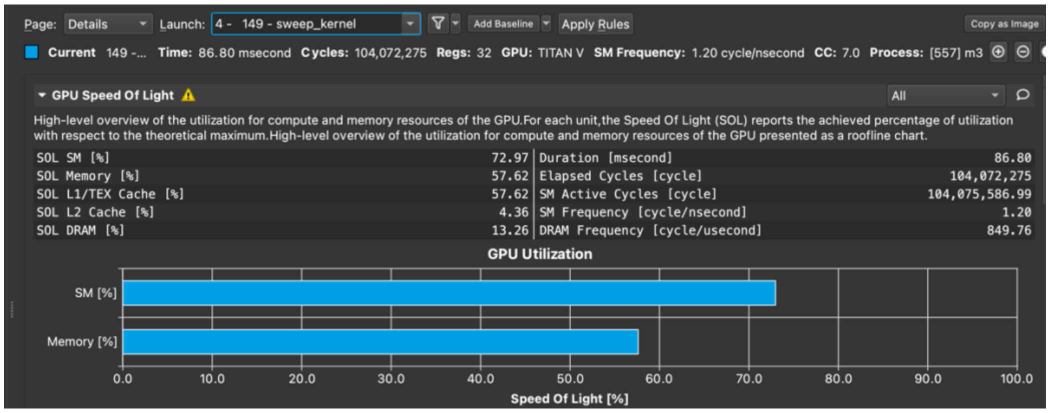
*This optimization would involve sweeping parameters like tile width and block size which would result in more threads per block increasing parallelism or more elements in a single tiled matrix multiplication operation. This is also a standalone optimization. For this particular optimization different tile widths like 16,32 and 64 were tried. The tile width of 16 provided the best performance. Moreover, the order of the thread parameters (threadIdx.x and threadIdx.y) was also swapped to test the different in the different kind of access pattern.*

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 | *0.225 ms* | *1.008 ms* | *4.401 ms* | *0.86* |
| 1000 | *2.182 ms* | *8.894 ms* | *10.284 ms* | *0.886* |
| 10000 | *15.97 ms* | *62.2325 ms* | *42.99 s* | *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 can be seen from the reduction in OP times and execution rimes that the optimization significantly improved performance for all datasets while retaining accuracy. The SOL utilization screenshot also shows that the SM usage % has increased from around 40% to around 70% for both kernels and the memory usage % has decreased from around 95% to 55%, indicating that performance was improved when parameters were sweeped. I believe the primary change that improved the performance thought was swapping the order of thread access in x and y.*



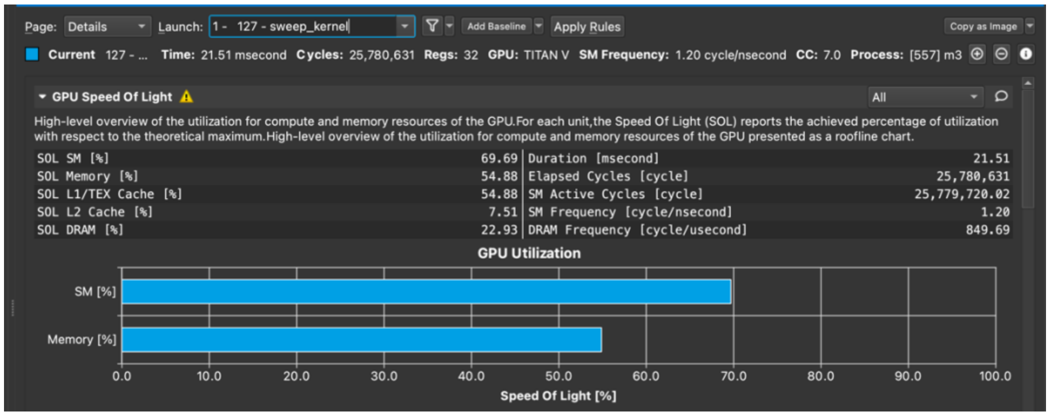
Current 149 -... Time: 86.80 msecond Cycles: 104,072,275 Regs: 32 GPU: TITAN V SM Frequency: 1.20 cycle/nsecond CC: 7.0 Process: [557] m3

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.High-level overview of the utilization for compute and memory resources of the GPU presented as a roofline chart.

```
SOL SM [%]                  72.97 | Duration [msecond]                    86.80
SOL Memory [%]              57.62 | Elapsed Cycles [cycle]          104,072,275
SOL L1/TEX Cache [%]        57.62 | SM Active Cycles [cycle]     104,075,586.99
SOL L2 Cache [%]             4.36 | SM Frequency [cycle/nsecond]           1.20
SOL DRAM [%]                13.26 | DRAM Frequency [cycle/usecond]       849.76
```

GPU Utilization



Current 127 - ... Time: 21.51 msecond Cycles: 25,780,631 Regs: 32 GPU: TITAN V SM Frequency: 1.20 cycle/nsecond CC: 7.0 Process: [557] m3

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.High-level overview of the utilization for compute and memory resources of the GPU presented as a roofline chart.

```
SOL SM [%]                  69.69 | Duration [msecond]                    21.51
SOL Memory [%]              54.88 | Elapsed Cycles [cycle]           25,780,631
SOL L1/TEX Cache [%]        54.88 | SM Active Cycles [cycle]      25,779,720.02
SOL L2 Cache [%]             7.51 | SM Frequency [cycle/nsecond]           1.20
SOL DRAM [%]                22.93 | DRAM Frequency [cycle/usecond]       849.69
```

GPU Utilization

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

*ECE 408 Lecture Slides*

5. **Optimization 5: Multiple kernel implementations + sweeping kernel**

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

*This technique uses the optimization 4 code for the kernels. I will use the kernel with the sweeped parameters from optimization 4 since it gave the best performance, I thought it would be best to optimize that even further.*

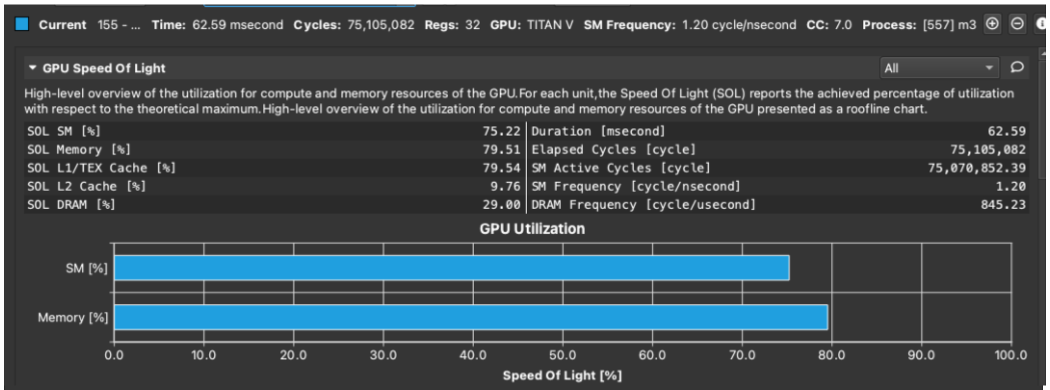
b. How does the optimization work? Did you think the optimization would increase performance of the forward convolution? Why? Does the optimization synergize with any of your previous optimizations?

*Two different kernels will be implemented here where the code from optimization 4 will be used in both but their tile sizes will be different, and both of these tile sizes will be multiples of the number of feature maps in the two layers. This synergizes with optimization 4 and should improve performance from there.*

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 | *1.275 ms* | *6.66 ms* | *2.846 s* | *0.86* |
| 1000 | *1.63 ms* | *6.25 ms* | *9.932 s* | *0.886* |
| 10000 | *14.54 ms* | *60.3 ms* | *39.841 s* | *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).

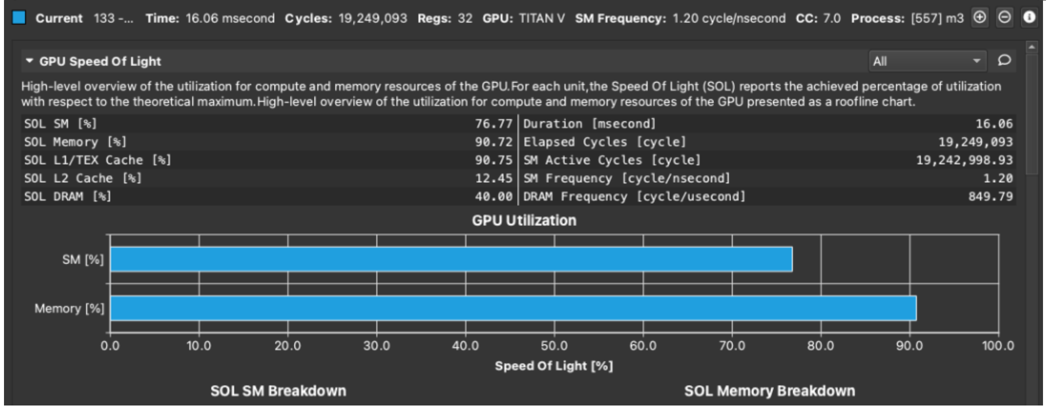


Current 155 - ... Time: 62.59 msecond Cycles: 75,105,082 Regs: 32 GPU: TITAN V SM Frequency: 1.20 cycle/nsecond CC: 7.0 Process: [557] m3

▾ 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.High-level overview of the utilization for compute and memory resources of the GPU presented as a roofline chart.

```
SOL SM [%]                 75.22 | Duration [msecond]                 62.59
SOL Memory [%]             79.51 | Elapsed Cycles [cycle]        75,105,082
SOL L1/TEX Cache [%]       79.54 | SM Active Cycles [cycle]   75,070,852.39
SOL L2 Cache [%]            9.76 | SM Frequency [cycle/nsecond]        1.20
SOL DRAM [%]               29.00 | DRAM Frequency [cycle/usecond]    845.23
```



Current 133 -... Time: 16.06 msecond Cycles: 19,249,093 Regs: 32 GPU: TITAN V SM Frequency: 1.20 cycle/nsecond CC: 7.0 Process: [557] m3

▾ 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.High-level overview of the utilization for compute and memory resources of the GPU presented as a roofline chart.

```
SOL SM [%]                 76.77 | Duration [msecond]                 16.06
SOL Memory [%]             90.72 | Elapsed Cycles [cycle]        19,249,093
SOL L1/TEX Cache [%]       90.75 | SM Active Cycles [cycle]   19,242,998.93
SOL L2 Cache [%]           12.45 | SM Frequency [cycle/nsecond]        1.20
SOL DRAM [%]               40.00 | DRAM Frequency [cycle/usecond]    849.79
```

*There is a slight improvement in performance in the the batch sizes of 1000, and 10000 from optimization 4 as the OP times have reduced a little due to the condition of account for the two layers. The performance however got worse in the 100-batch size possibly due to the condition statements possibly causing warp divergence.*

*It can be seen from the SOL screenshot however that there is greater memory usage % than optimization 4 due to the conditional statements possibly. However, the SM % increased a bit resulting in better performance I believe since the two kernels had the two layers applied to them separately which leads to a slight reduction in time for the second layer leading to an overall shorter time consumed.*

*This optimization results in performance closest to < 70 ms out of all the optimizations I attempted.*

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

*ECE 408 Lecture Slides + Campuswire posts*