Name: Joseph Thomas

NetID: *jpt7* 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 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	0.198917ms	0.585584ms	0m2.964s	0.86
1000	2.77284ms	6.33284ms	0m12.430s	0.886
5000	14.748ms	32.8914ms	0m51.459s	0.871

me(%)	Total Time	Calls	Average	Minimum	Maximum	Name
69.1	576325657	20	28816282.9	36210	285361046	cudaMemcpy
23.8	198959200		9947960.0	2913		cudaMalloc
4.9			4066864.1	4007		cudaDeviceSynchronize
1.9			1581031.2	19018		cudaLaunchKernel
0.3	2810686	20	140534.3	2102	580987	cudaFree
nerating	g CUDA Kernel Si g CUDA Memory Op el Statistics (peration Stati	stics			
			Average	Minimum	Maximum	Name
100.0	40643429	6	6773904.8	7360	32017498	conv forward kernel
0.0			1408.0	1376	1440	
0.0	2592	2	1296.0	1280	1312	prefn_marker_kernel
DA Memor	ry Operation Sta	atistics (nano	seconds)			
me(%)	Total Time	Operations	Average	Minimum	Maximum	Name
			85458960.7	23328	284642563	[CUDA memcpy DtoH]
91.7	512753764	6		23320		
				1152		[CUDA memcpy HtoD]
91.7 8.3		14				[CUDA memcpy HtoD]
91.7 8.3	46333124	14			24003756	
91.7 8.3 DA Memor	46333124 ry Operation Sta	14 atistics (KiB) perations	33 0 9508.9 Average	1152 Minimum	24003756 N	laximum Name
91.7 8.3 DA Memor	46333124	14 atistics (KiB)	3309508.9	1152	24003756 N	

Optimization 1: Weight matrix (kernel values) in constant memory (3 point)

1.

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

I chose to implement weight matrix (kernel values) in constant memory because this technique allows a quicker access to constant memory comparing to accessing from 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?

This implementation works by making constant memory copy the host mask and make it readily available for blocks to access rather than blocks accessing global memory like it used to in M2 implementation. I think the optimization would work assuming that accessing constant memory is quicker than accessing global memory. No, this optimization doesn't synergizes with any other 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	0.172613 ms	0.587303 ms	0m1.669s	0.86
1000	1.56391 ms	5.70719 ms	0m10.278s	0.886
5000	7.70291 ms	28.9042 ms	49.097s	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).

Yes, it was successful because it is quicker to access constant memory than global memory. While comparing the nsys of M2 and optimization, it is observable that cudaMemcpy and cudaMemcpyToSymbol combined time of the optimization is comparably below cudaMemcpy time of M2. It is also observable that this difference has also effect on kernel time, giving assurance that the optimization is successful.

Nsys:

Total Time	Calls	Average	Minimum	Maximum	Name	
541865144	20	27093257.2	26228	290676856	cudaMemcpy	
193600378	20	9680018.9	2835	190439130	cudaMalloc	
		3693647.4	3957			
		1635122.0				1
13/031	Ь	22838.5	18386	25909	cudaMemcpy losyn	ipoT
g CUDA Kernel S	Statistics					
g CUDA Memory (Operation Stat	tistics				
		130103111				
Total Time	Instances	Average	Minimum	Maximum	Name	
		6151997.3	6944	29051025		
		1424.0	1408			
2528	2	1264.0	1216	1312	prefn_marker_ke	ernel
ry Operation St	tatistics (nar	noseconds)				
Total Time	Operations	Average	Minimum	Maximum	Name	
489429003	6	81571500.5	23200	289901555	[CUDA memcpy Dt	coH]
		1824.0	1536			
ry Operation St	tatistics (KiB	3)				
Total C	Operations	Average	Minimu	um Ma	aximum Name	
862672.0		143778.7	148.53	35 500	0000.0 [CUDA me	mcpy Dto
	14	19729.0	0.00	34 144	4453.0 [CUDA me	mcpv Hto
276206.0	14	13723.0				
	Total Time 541865144 193600378 36936474 16351220 2452784 137031 g CUDA Kernel S g CUDA Memory C el Statistics (Total Time 36911984 2528 ry Operation St Total Time 489429003 4589544 ry Operation St Total Time	193600378 20 36936474 10 16351220 10 2452784 20 137031 6 g CUDA Kernel Statistics g CUDA Memory Operation Statel Statistics (nanoseconds) Total Time Instances 36911984 6 2848 2 2528 2 ry Operation Statistics (nan Total Time Operations 489429003 6 45895432 14 10944 6 ry Operation Statistics (KiB Total Operations	Total Time Calls Average 541865144 20 27093257.2 193600378 20 9680018.9 36936474 10 3693647.4 16351220 10 1635122.0 2452784 20 122639.2 137031 6 22838.5 g CUDA Kernel Statistics g CUDA Memory Operation Statistics el Statistics (nanoseconds) Total Time Instances Average 36911984 6 6151997.3 2848 2 1424.0 2528 2 1264.0 ry Operation Statistics (nanoseconds) Total Time Operations Average 489429003 6 81571500.5 45895432 14 3278245.1 10944 6 1824.0 ry Operation Statistics (KiB) Total Operations Average	Total Time Calls Average Minimum 541865144 20 27093257.2 26228 193600378 20 9680018.9 2835 36936474 10 3693647.4 3957 16351220 10 1635122.0 9601 2452784 20 122639.2 3615 137031 6 22838.5 18386 g CUDA Kernel Statistics g CUDA Memory Operation Statistics el Statistics (nanoseconds) Total Time Instances Average Minimum 36911984 6 6151997.3 6944 2848 2 1424.0 1408 2528 2 1264.0 1216 ry Operation Statistics (nanoseconds) Total Time Operations Average Minimum 489429003 6 81571500.5 23200 45895432 14 3278245.1 1152 10944 6 1824.0 1536 ry Operation Statistics (KiB) Total Operations Average Minimum 489429003 A 81571500.5 23200 45895432 14 3278245.1 1152 10944 6 1824.0 1536 ry Operation Statistics (KiB) Total Operations Average Minimum	Total Time Calls Average Minimum Maximum 541865144 20 27093257.2 26228 290670856 193600378 20 9680018.9 2835 190439130 36936474 10 3693647.4 3957 29051955 16351220 10 1635122.0 9601 16201442 2452784 20 122639.2 3615 627378 137031 6 22838.5 18386 25909 g CUDA Kernel Statistics g CUDA Memory Operation Statistics el Statistics (nanoseconds) Total Time Instances Average Minimum Maximum 36911984 6 6151997.3 6944 29051025 2848 2 1424.0 1408 1440 2528 2 1264.0 1216 1312 ry Operation Statistics (nanoseconds) Total Time Operations Average Minimum Maximum 489429003 6 81571500.5 23200 289901555 45895432 14 3278245.1 1152 23991995 10944 6 1824.0 1536 2240 ry Operation Statistics (KiB) Total Operations Average Minimum Maximum Total Operations Average Minimum Maximum Average Minimum Maximum 489429003 6 81571500.5 23200 289901555 45895432 14 3278245.1 1152 23991995 10944 6 1824.0 1536 2240 ry Operation Statistics (KiB) Total Operations Average Minimum Maximum Total Operations Average Minimum Maximum	Total Time

Nsight-Compute:



e. What references did you use when implementing this technique? Lecture slides and Textbook

2. Optimization 2: Input channel reduction: tree

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

I chose Input channel reduction: tree as my optimization because input reduction tree increase parallelism.

- 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?

 Optimization works by using tree reduction to increase parallelism by assigning each thread to a channel and then making use of reduction method to achieve higher parallelism. I think the optimization would increase performance of the forward convolution because it manages many channels at same 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	0.205646 ms	0.678121 ms	0m1.581s	0.86
1000	1.89966 ms	6.63198 ms	0m10.352s	0.886
5000	9.42865 ms	33.073 ms	Om51.558s	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).

No, it was not successful and it is comparably worse than M2 baseline. I think the reason could be because of the use of syncthreads, which is observable when you compare the difference in kernel time shown in nsys.

Nsys:

me(%)	Total Time	Calls	Average	Minimum	Maximum	Name
70.2 21.6	557524386	20 20	27876219.3 8591542.2	32290		cudaMemcpy cudaMalloc
5.4	171830843 42560390	10	4256039.0	2890 4072		cudaDeviceSynchronize
2.5	19512742	10	1951274.2	15772		cudaLaunchKernel
0.3	2701858	20	135092.9	2356		cudaFree
nerating	g CUDA Kernel Sta	atistics				
nerating	g CUDA Memory Op	eration Stati	stics			
OA Kerne	el Statistics (na	anoseconds)				
ne(%)	Total Time	Instances	Average	Minimum	Maximum	Name
100.0	42529424		7088237.3	6784		conv_forward_kernel
0.0	9280	2	4640.0	1312	7968	prefn_marker_kernel
100.0					7968	prefn_marker_kernel
0.0 0.0	9280	2 2	4640.0 1440.0	1312	7968	prefn_marker_kernel
0.0 0.0 0.0	9280 2880 ry Operation Sta	2 2 tistics (nano	4640.0 1440.0	1312	7968	prefn_marker_kernel do_not_remove_this_kern
DA Memor	9280 2880 ry Operation Sta Total Time (2 2 tistics (nano Operations 	4640.0 1440.0 seconds) Average	1312 1408 Minimum 12640	7968 1472 Maximum 299180270	prefn_marker_kernel do_not_remove_this_kern Name [CUDA memcpy DtoH]
0.0 0.0 0.0 0.0 0.0	9280 2880 ry Operation Sta	2 2 tistics (nano Operations	4640.0 1440.0 (seconds) Average	1312 1408 Minimum 12640	7968 1472 Maximum 299180270	prefn_marker_kernel do_not_remove_this_kern Name
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0	9280 2880 ry Operation Sta Total Time (2 2 tistics (nano Operations 	4640.0 1440.0 seconds) Average 	1312 1408 Minimum 12640	7968 1472 Maximum 299180270	prefn_marker_kernel do_not_remove_this_kern Name [CUDA memcpy DtoH]
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0	9280 2880 ry Operation Star Total Time 6 499937365 39633990 ry Operation Star	2 2 tistics (nano Operations 	4640.0 1440.0 seconds) Average 	1312 1408 Minimum 12640	7968 1472 Maximum 	prefn_marker_kernel do_not_remove_this_kern Name [CUDA memcpy DtoH]
100.0 0.0 0.0 0.0 0.0 DA Memori 7.3	9280 2880 ry Operation Star Total Time 6 499937365 39633990 ry Operation Star	2 2 tistics (nano Operations 6 14	4640.0 1440.0 seconds) Average 	1312 1408 Minimum 12640 1152	7968 1472 Maximum 299180270 20964875	prefn_marker_kernel do_not_remove_this_kerne Name [CUDA memcpy DtoH] [CUDA memcpy HtoD]

Nsight Compute:



e. What references did you use when implementing this technique? Lecture slides and Textbook

- 3. Optimization 3: Multiple kernel implementations for different layer sizes (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.

I chose Multiple kernel implementation as my optimization because I think that kernel incorporating different layer sizes will improve 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?

It has two separate kernels with the same algorithm but optimized to incorporate different layer sizes. I think that this optimization would increase performance of the forward convolution because this implementation modifies its parameters to incorporate unique inputs. This optimization synergizes with Weight matrix (kernel values) in constant memory.

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	0.176561 ms	0.586961 ms	0m4.156s	0.86
1000	1.5644 ms	5.70981 ms	0m11.248s	0.886
5000	7.7109 ms	28.9908 ms	0m52.801s	0.871

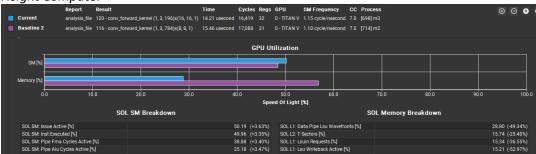
d. Was implementing this optimization successful in improving performance? Why or why not? Include profiling results from *nsys* and *Nsight-Compute* to justify your answer, directly comparing to your baseline (or the previous optimization this one is built off of).

Yes, implementing this optimization was successful in improving performance because we successfully incorporated parameters to match with specific unique inputs. It is observable in nsys that the kernel time has a significant improvement when compared to M2 baseline and Weight matrix (kernel values) in constant memory optimization which was synergized with this optimization.

Nsvs:

	Total	Operations	Average	Minimum	Ma	aximum	Name
	10001	oper actoris	Average	HIHIM	716	ATIIII	Name
	862672.0	6	143778.7	148.535	506	9000.0	[CUDA memcpy Dto
	276206.0	14	19729.0	0.004	144	1453.0	[CUDA memcpy Hto
	14.0	6	2.4	0.316		12.0	[CUDA memcpy Dto
	g CUDA API Sta Statistics (na						
ne(%)	Total Time	Calls	Average	Minimum	Maximum	Name	
70.0	550587199	20	27529359.7	28116	304621197	cudaMe	emcDV
22.7	178988056	20	8949402.8	2598	175356525		
4.7	36800454	10	3680045.4	3497			eviceSynchronize
2.2	17638187	10	1763818.7	10117	17478125	cudaLa	aunchKernel
0.3	2700026	20	135001.3	3406	656955	cudaFi	ree
0.0	463344						
0.0	162344	6	27057.3	24230	29098	cudaMe	emcpyToSymbol
neratin neratin DA Kern	g CUDA Kernel g CUDA Memory el Statistics	Statistics Operation Stat: (nanoseconds)		24230 Minimum	29098 Maximum		emcpyToSymbol
neratin neratin DA Kern ne(%)	g CUDA Kernel g CUDA Memory el Statistics	Statistics Operation Stat: (nanoseconds) : Instances	stics	Minimum	Maximum	Name	emcpyToSymbol
neratin neratin DA Kern ne(%)	g CUDA Kernel g CUDA Memory el Statistics Total Time	Statistics Operation Stat: (nanoseconds) : Instances	stics Average		Maximum 28965795	Name conv_r	 Forward_kernel
meratin meratin DA Kern me(%)	g CUDA Kernel g CUDA Memory el Statistics Total Time	Statistics Operation Stat: (nanoseconds) Instances 6 2	Average 	Minimum	Maximum 28965795 1472	Name conv_r do_not	Forward_kernel
meratin DA Kern Me(%) 100.0 0.0 0.0	g CUDA Kernel g CUDA Memory el Statistics Total Time 36779600 2917 2560	Statistics Operation Stat: (nanoseconds) Instances 6 2	Average 	Minimum 7040 1440	Maximum 28965795 1472	Name conv_r do_not	Forward_kernel t_remove_this_ker
neratin neratin DA Kern me(%) 100.0 0.0 0.0	g CUDA Kernel g CUDA Memory el Statistics Total Time 36779606 2912 2566 ry Operation S	Statistics Operation Stat: (nanoseconds) Instances 6 2 2	Average 6129933.3 1456.0 1280.0	Minimum 7040 1440	Maximum 28965795 1472	Name conv_f do_not prefn_	Forward_kernel t_remove_this_ker
neratin neratin DA Kern me(%) 100.0 0.0 0.0	g CUDA Kernel g CUDA Memory el Statistics Total Time 36779606 2912 2566 ry Operation S	Statistics Operation Stat: (nanoseconds) Instances 2 2 Statistics (nanoseconds)	Average 6129933.3 1456.0 1280.0	Minimum 7040 1440 1248	Maximum 28965795 1472 1312 Maximum	Name conv_do_not prefn_	forward_kernel t_remove_this_ker marker_kernel
neratin DA Kern me(%) 100.0 0.0 0.0 DA Memo me(%)	g CUDA Kernel g CUDA Memory el Statistics Total Time 36779606 2912 2560 ry Operation S	Statistics Operation Stat: (nanoseconds) Instances 2 2 Statistics (nanoseconds)	Average 6129933.3 1456.0 1280.0 0seconds) Average	Minimum 7040 1440 1248 Minimum	Maximum 	Name conv_do_not prefn_ Name CUDA	Forward_kernel t_remove_this_ker

Nsight Compute:



e. What references did you use when implementing this technique? Lecture slides and Textbook

4. Optimization 4: Tuning with restrict and loop unrolling (3 points)

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

I chose to implement Tuning with restrict and loop unrolling as my optimization because restrict and unrolling improves performance by implementing a single iteration rather than multiple iteration.

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?

In the implementation every data pointer is changed to restricted pointers. A loop is then introduced to iterate through elements of the matrices to allow an advantage to small K values. I think that this optimization would increase performance of the forward convolution because restrict and unrolling improves performance by implementing a single iteration rather than multiple iteration. This optimization don't synergize with any of the other 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	0.215549	0.572844	0m1.777s	0.86
	ms	ms		
1000	2.02565 ms	5.59096 ms	0m10.812s	0.886
5000	10.036 ms	27.8985 ms	0m51.108s	0.871

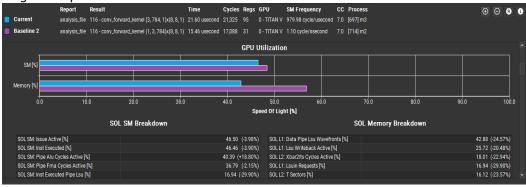
d. Was implementing this optimization successful in improving performance? Why or why not? Include profiling results from *nsys* and *Nsight-Compute* to justify your answer, directly comparing to your baseline (or the previous optimization this one is built off of).

This optimization was successful in improving performance because restrict and unrolling has increased performance by implementing a single iteration rather than multiple iteration. It is observable in nsys that both cudaMemcpy and cudaMalloc times are significantly lower while comparing to baseline M2. Which is also reflected in the difference in kernel execution time. It is also observable in the Nsight compute that there is a major difference in Memory(%) indicating it is running efficiently than M2 baseline.

Nsvs:

ivsys:						
	CUDA API Stat tatistics (nar					
Time(%)	Total Time	Calls	Average	Minimum	Maximum	Name
70.2 22.4 4.9	544964030 173957519 38317121	20	27248201.5 8697875.9 2394820.1	28501 2824 812	170617523	cudaMemcpy cudaMalloc cudaDeviceSynchronize
2.2	16992043 2323208	10		15553 2758	16790758	cudaLaunchKernel cudaFree
	CUDA Kernel S					
	CUDA Memory (l Statistics (tistics			
Time(%)	Total Time	Instances	Average	Minimum	Maximum	Name
100.0 0.0 0.0	38172809 2816 2720	2		7008 1376 1344	27928627 1440 1376	
CUDA Memor	y Operation St	atistics (na	noseconds)			
Time(%)	Total Time	Operations	Average	Minimum	Maximum	Name
92.5 7.5	498370400 40201947			12608 1152		[CUDA memcpy DtoH] [CUDA memcpy HtoD]
CUDA Memor	y Operation St	atistics (Ki	В)			
	Total ()perations	Average	Minimu	m M	laximum Name
	62672.0	6	143778.7	148.53	5 50	00000.0 [CUDA memcpy DtoH]
2	76206.0	14	19729.0	0.00	4 14	4453.0 [CUDA memcpy HtoD]





e. What references did you use when implementing this technique? Lecture and Textbook 5. **Optimization 5:** Tiled shared memory convolution (2 points)

(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.

I chose Tiled shared memory convolution as my optimization technique because using shared memory will reduce the time consumed for accessing input features.

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?

We use shared memory to improve memory reuse rather than accessing global memory which tends to be slow. I think that this optimization would increase performance of the forward convolution, because it is quicker to access shared memory than global memory. This optimization doesn't synergize with any other 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	0.751419 ms	1.19764 ms	0m2.830s	0.86
1000	2.54745 ms	6.93911 ms	0m11.514s	0.886
5000	11.7706 ms	32.3134 ms	0m52.301s	0.871

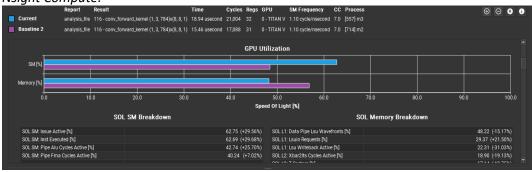
d. Was implementing this optimization successful in improving performance? Why or why not? Include profiling results from *nsys* and *Nsight-Compute* to justify your answer, directly comparing to your baseline (or the previous optimization this one is built off of).

No, the optimization was not successful in improving performance because there was not much reuse of the shared memory, which would have led to the misuse of resources in shared memory. It is also observable with increased kernel time while comparing it with M2 baseline. While it is evident also in Nsight Compute with the difference in SM(%) while comparing with M2 baseline.

Nsvs:

me(%)	Total Time	Calls	Average	Minimum	Maximum	Name
64.6	1926927985	20	96346399.3	2416	1921392706	cudaMalloc
19.3	576558759	20	28827937.9	29757		cudaMemcpy
	419920123	20	20996006.1	2575		
1.5 0.5	44363512 15830224	10 10	4436351.2 1583022.4	6336 23968		cudaDeviceSynchronize cudaLaunchKernel
	CUDA Kernel St					
	CUDA Memory Op 1 Statistics (n		stics			
ime(%)	Total Time	Instances	Average	Minimum	Maximum	Name
100.0	43134822		7189137.0	11936	31823519	conv_forward_kernel do not remove this kerne
0.0	2944					
		2	1472.0			
0.0			1376.0			do_not_remove_this_kerne prefn_marker_kernel
			1376.0			
UDA Memor	2752 y Operation Sta	2 ntistics (nano	1376.0	1312	1440	prefn_marker_kernel
UDA Memor	2752 y Operation Sta Total Time 525092701	2 ntistics (nano Operations 	1376.0 seconds) Average	1312 Minimum	1440 Maximum 308200198	prefn_marker_kernel Name [CUDA memory DtoH]
UDA Memor	2752 y Operation Sta Total Time	2 ntistics (nano Operations 	1376.0 seconds) Average	1312 Minimum	1440 Maximum 308200198	prefn_marker_kernel
UDA Memor ime(%) 91.8 8.2	2752 y Operation Sta Total Time 525092701	2 distics (nand Operations 6 14	1376.0 seconds) Average 	1312 Minimum	1440 Maximum 308200198	prefn_marker_kernel Name [CUDA memory DtoH]
UDA Memor ime(%) 91.8 8.2	y Operation Sta Total Time 525092701 46616500	2 otistics (nano Operations 6 14	1376.0 seconds) Average 87515450.2 3329750.0	1312 Minimum 23232 1120	Maximum 308200198 24690146	prefn_marker_kernel Name [CUDA memcpy DtoH] [CUDA memcpy HtoD]
UDA Memor ime(%) 91.8 8.2	2752 y Operation Sta Total Time 525092701 46616500	2 otistics (nano Operations 6 14	1376.0 seconds) Average 	1312 Minimum	Maximum 308200198 24690146	prefn_marker_kernel Name [CUDA memory DtoH]
UDA Memor ime(%) 91.8 8.2 UDA Memor	2752 Ty Operation Sta Total Time 525092701 46616500 Ty Operation Sta Total Op	2 otistics (name Operations 6 14 otistics (KiB) Derations	1376.0 seconds) Average 87515450.2 3329750.0	1312 Minimum 	Maximum 	prefn_marker_kernel Name [CUDA memcpy DtoH] [CUDA memcpy HtoD]
JDA Memor ime(%) 91.8 8.2 JDA Memor	y Operation Sta Total Time 525092701 46616500	2 otistics (nano Operations 6 14	1376.0 seconds) Average 87515450.2 3329750.0	1312 Minimum 	Maximum 308200198 24690146	prefn_marker_kernel Name [CUDA memcpy DtoH] [CUDA memcpy HtoD]





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

- 6. **Optimization 6:** Sweeping various parameters to find best (0.5 point) (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.

I chose Sweeping various parameters as my optimization because it needs very little changes to the code and can result to improvement in 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?
 - For this optimization I tested Tile Sizes and fixed on Tile Size of 24. I think that the optimization would increase performance of the forward convolution because trying different Tile Size can result to a better performance. This optimization doesn't synergize with any of the other 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).

			Total	
Batch Size	Op Time 1	Op Time 2	Execution	Accuracy
			Time	
100	0.243166 ms	0.687001	0m1.658s	0.86
		ms		
1000	2.29288 ms	6.77237 ms	0m10.609s	0.886
5000	11.337 ms	34.2141 ms	0m51.766s	0.871

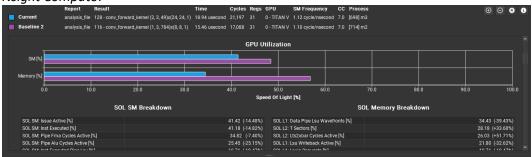
d. Was implementing this optimization successful in improving performance? Why or why not? Include profiling results from *nsys* and *Nsight-Compute* to justify your answer, directly comparing to your baseline (or the previous optimization this one is built off of).

Implementing this optimization is successful in improving performance because as seen with Op time which has decreased significantly from M2 baseline. It is also observable in nsys where the kernel time has improved significantly show that the performance has improved. It is also represented in Nsight Compute where both SM(%) and Memory(%) are lower compared to M2 baseline.

Nsys:

Nsys:							
	CUDA API Stat tatistics (nan						
Time(%)	Total Time	Calls	Average	Minimum	Maximum	Name	
68.9	532208961	20	26610448.1	35631	284599990	cudaMemcpy	
22.7	175285321	20	8764266.1	3400		cudaMalloc	
5.9	45439970		4543997.0	3364	33885381	cudaDeviceSynchroni	ize
2.1	16397659	10	1639765.9	25356	16144342	cudaLaunchKernel	
0.4	2751679	20	137584.0	2706	616860	cudaFree	
Generating	CUDA Kernel S	tatistics					
Generating	CUDA Memory 0	peration Stat	istics				
	l Statistics (
Time(%)	Total Time		Average	Minimum	Maximum	Name	
100.0			7569121.8	8352	33883721	conv forward kernel	1
0.0	2816	2	1408.0	1376	1440	prefn marker kernel	l
0.0	2784	2	1392.0	1376	1408	do_not_remove_this	_kernel
CUDA Memor	y Operation St	atistics (nan	noseconds)				
Time(%)	Total Time	Operations	Average	Minimum	Maximum	Name	
91.3	482445604	6	80407600.7	23232	283893816	[CUDA memcpy DtoH]	
8.7	45906150	14	3279010.7	1152	23995517	[CUDA memcpy HtoD]	
CUDA Memory	y Operation St	atistics (KiB	3)				
	Total 0	perations	Average	Minim	num P	laximum Name	
	62672 0		143770 7	140.5	35	ACCUPA ACCUPA	, DtoHi
81	62672.0	6	143//8./	148.5	55 56	00000.0 [CUDA memcpy	у БСОН]
2	76206.0	14	19729.0	0.0	004 14	4453.0 [CUDA memcpy	/ HtoD]

Nsight Compute:



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