Goal:
Use NVIDIA's CUDA library to grayscale a JPEG image. The block size times the grid size had to be the size of the image where each pixel was modified using the luminosity method of

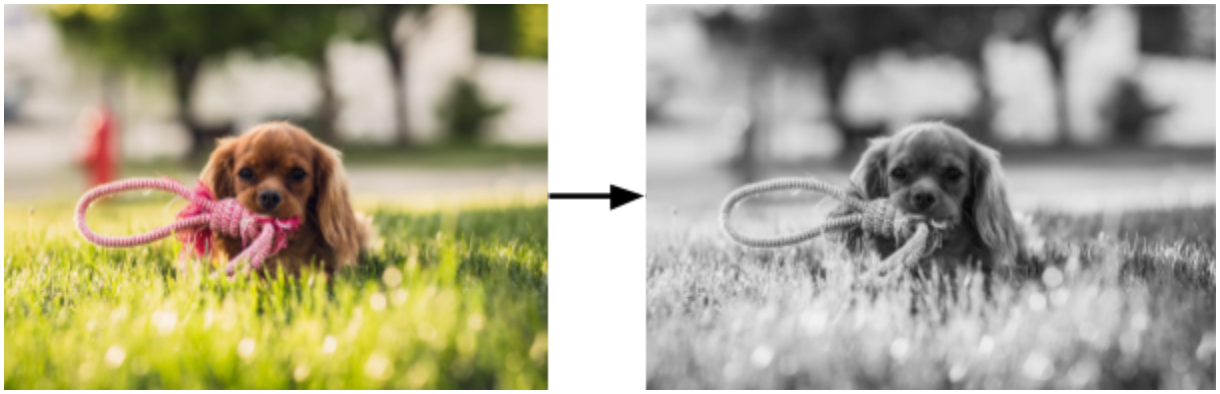
$$red[index] * 0.3 + green[index] * 0.59 + blue[index] * 0.11$$

Code:
PyCuda module was used as an interface with the CUDA library. The kernel function was written in C and ported into the SourceModule class to be interpreted and used. By using PyCuda we could use the Pillow Image processing module and auto initialize the CUDA backend. The uchar4 array was created from the image and separated by the channel dimensions: red, green, and blue. Using the image shape and the block dimension input, we can then calculate the grid dimensions (image area = block area * grid area). For example, the puppy image shown below had an image area of 12252246 and with a dim_block input of 16 (<16,16,1> which is 256 threads) we are left with a possible grid size of 47972. The transformed array using the luminosity method was then saved as a JPEG image.

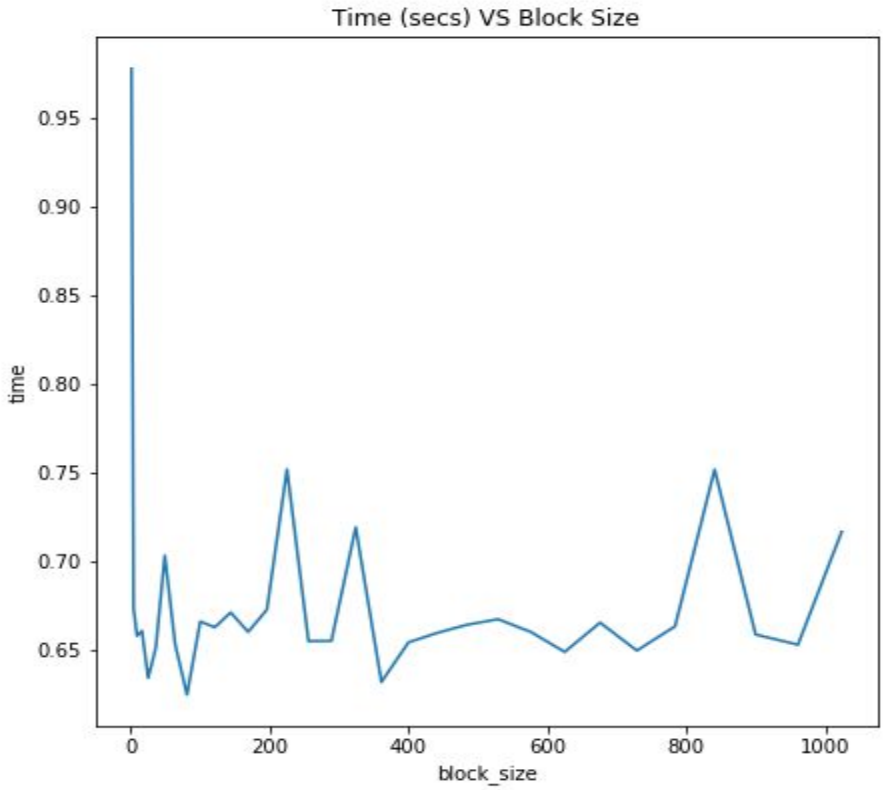
Results:
The time variations were not as exciting as I hoped. Processing the 2Mb image took a quick 1.2 seconds with a block size of 1 x 1 and .65 seconds for a block size of 2 x 2. From there the execution time of the grayscale filter was still an impressive ~.65 to ~.7 seconds until 1024 where CUDA version 10 has been hard-capped.

Conclusion:
There is a balance of multithreading for the GPU as it is on the CPU. If you don't use the threads they will be inactivate and could have been used to increase runtime. However, you can get to a saturation point and a structural limit with hardware. Here the image was able to be chewed through with only a block dimension of 2 x 2 just as fast as a block dimension of 32 x 32. There is an overhead that comes with multithreading that the GPU cannot currently escape from either the shared memory and register usage. This assignment was a good example of what it means to optimize and choose threads wisely.

Example out.jpg from my grayscale code:



Time VS Block Size: Starting at 1 and ending at 1024



Complete Data Results:

	image_size	block_size	grid_size	time
0	12252246	1	12252246	0.97831
1	12252246	4	3063776	0.67303
2	12252246	9	1361837	0.65749
3	12252246	16	766480	0.66041
4	12252246	25	490776	0.63373
5	12252246	36	341055	0.65073
6	12252246	49	250717	0.70296
7	12252246	64	191888	0.65247
8	12252246	81	151686	0.62426
9	12252246	100	122694	0.66564
10	12252246	121	101400	0.66238
11	12252246	144	85562	0.67078
12	12252246	169	72600	0.65977
13	12252246	196	62935	0.67234
14	12252246	225	54626	0.75174
15	12252246	256	47972	0.65452
16	12252246	289	42757	0.65471
17	12252246	324	38001	0.71889
18	12252246	361	34126	0.63145
19	12252246	400	30745	0.65385
20	12252246	441	28085	0.65917
21	12252246	484	25350	0.66377
22	12252246	529	23375	0.66696
23	12252246	576	21480	0.65979
24	12252246	625	19780	0.64846
25	12252246	676	18150	0.66502
26	12252246	729	16854	0.64922
27	12252246	784	15862	0.66295
28	12252246	841	14652	0.75161
29	12252246	900	13728	0.65830
30	12252246	961	12927	0.65255
31	12252246	1024	12060	0.71640

Top 10 Results Based On Time:

	image_size	block_size	grid_size	time
8	12252246	81	151686	0.62426
18	12252246	361	34126	0.63145
4	12252246	25	490776	0.63373
24	12252246	625	19780	0.64846
26	12252246	729	16854	0.64922
5	12252246	36	341055	0.65073
7	12252246	64	191888	0.65247
30	12252246	961	12927	0.65255
19	12252246	400	30745	0.65385
15	12252246	256	47972	0.65452