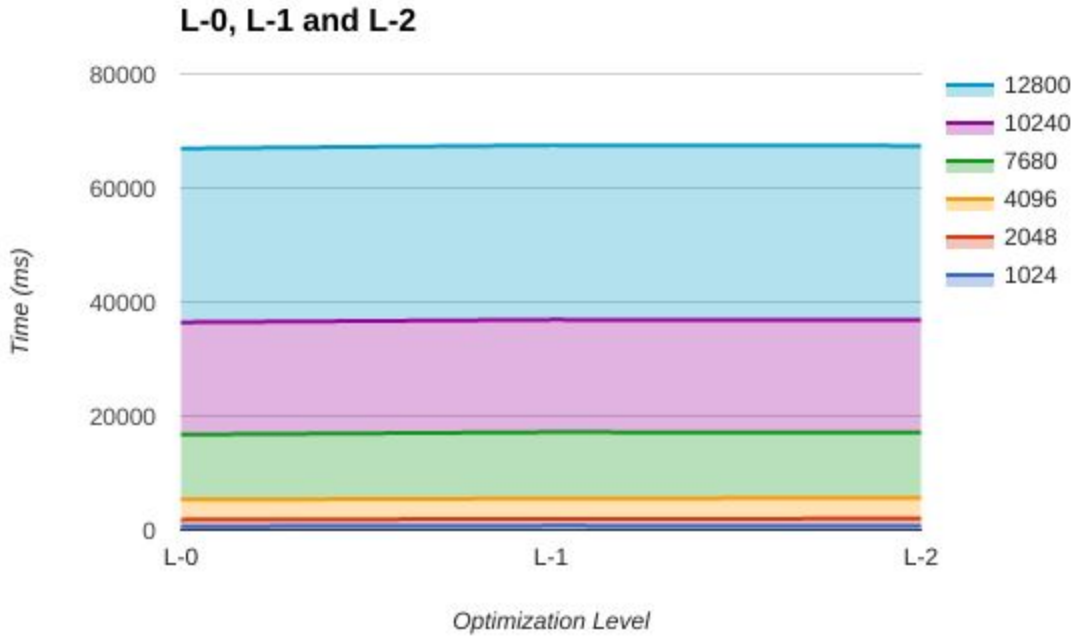
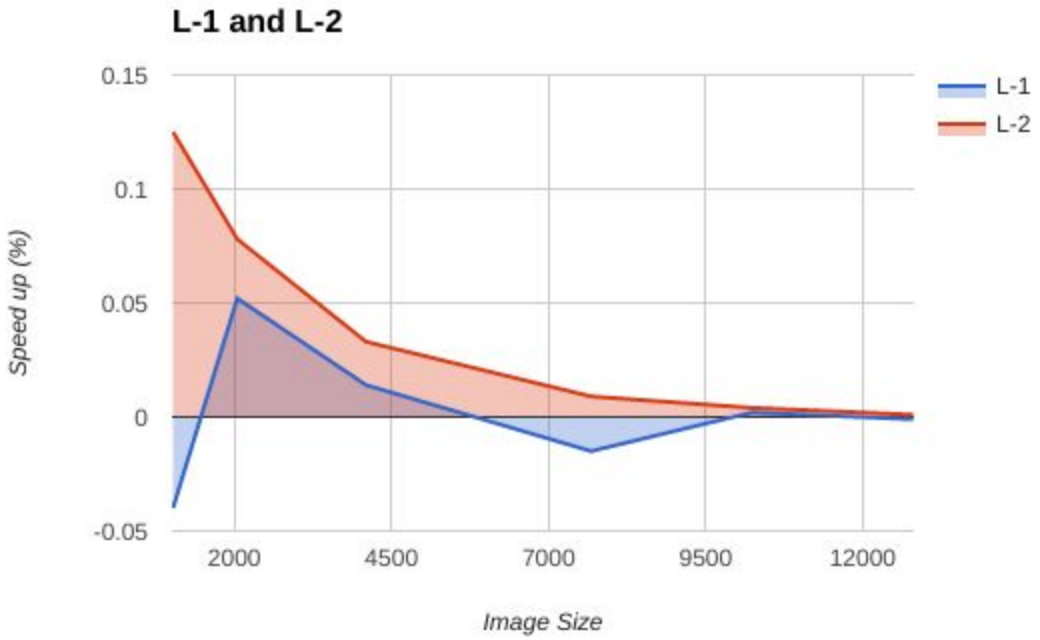
Cameron Franke
4/22/17
HPC

Program 5: GPGPU

	L-0	L-1	L-2
1024	629	748	719
2048	1167	1199	1266
4096	3510	3581	3633
7680	11434	11636	11454
10240	19691	19722	19774
12800	30507	30614	30558





For this project I used two approaches in an attempt to improve the performance of the GPU optimized convolutional code. The my “L 0” optimization attempt relied on shared memory and caching to achieve a speedup. By writing cache friendly code we can improve execution speed. The idea with this optimization is that accessed to main memory for any given pixel will only happen once because after the information has been transferred to the GPU device it will reside in the L2 cache until the caching algorithm decides that it is no longer needed. Since the only calls to main memory in this program will be in reference to the image, it is highly likely that the needed information will be in the L2 cache after another thread has requested it. This is a well established optimization but as you can see from the graph above both my L1 and L2 version saw diminishing speedup as the image size increased. This is because the size of the images started to exceed the size of the GPU L2 cache. My second optimization was to compile and run a similar version of my L-1 program with the “ffast math” compiler tag. This compiler optimization allows for some non ieee optimizations to be performed. The results with this optimization were slightly better than that of my L-1 program. As you can see in the image above, the L-2 optimization achieves a slightly higher speedup.

It is worth mentioning that the data above was collected after only 10 test for each data point. This is not enough to be scientifically sound, especially on a cluster shared by so many students. There is some inconsistency in the results that is likely due to varying workloads.