Cameron Calv and Nicholas Sica
ECEC 413 Intro to Parallel Computer Architecture
Assignment 5: Separable Convolution (CUDA)

Parallelization

The code was parallelized by allowing threads that execute on the GPU device to calculate various results of the convolution operation using data available by the thread and block information. The optimized version of the implementation makes calculations without needing to copy over the information of the kernel to the device memory. Further optimization was done by using shared memory over global memory to really enhance performance.

Evaluating Parallelization

Below are tables that showcase the execution time and speed-up of the program when run on different data sizes with values plotted with respect to the thread block sizes.

Table 1: Evaluating the execution time of the program on Drexel COE college's Xunil server.

	Block Size:	4	8	16	32
Data Size	Type				
2048	Gold	0.374152	0.355329	0.343798	0.408905
	Naïve	0.009519	0.006333	0.003573	0.003211
	Optimized	0.00182	0.001444	0.00066	0.000619
	Opt Shared	0.000877	0.000512	0.000426	0.000641
4096	Gold	1.291276	1.315384	1.265755	1.30274
	Naïve	0.037682	0.019153	0.015826	0.012346
	Optimized	0.007902	0.004215	0.003094	0.002248
	Opt Shared	0.003214	0.00202	0.001608	0.002127
8192	Gold	5.156335	5.164781	5.203758	5.157842
	Naïve	0.132175	0.088599	0.055321	0.056512
	Optimized	0.028537	0.016891	0.009252	0.00825
	Opt Shared	0.011974	0.006878	0.0056	0.007381

Execution Time

Table 2: Evaluating the speed-up of the program with parallel threads on Drexel COE college's Xunil server.

	Block Size:	4	8	16	32
Data Size	Type				
2048	Naïve	3931%	5611%	9622%	12735%
	Optimized	20558%	24607%	52091%	66059%
	Opt Shared	42663%	69400%	80704%	63792%
4096	Naïve	3427%	6868%	7998%	10552%
	Optimized	16341%	31207%	40910%	57951%
	Opt Shared	40177%	65118%	78716%	61248%
8192	Naïve	3901%	5829%	9406%	9127%
	Optimized	18069%	30577%	56245%	62519%
	Opt Shared	43063%	75091%	92924%	69880%

Speed-Up

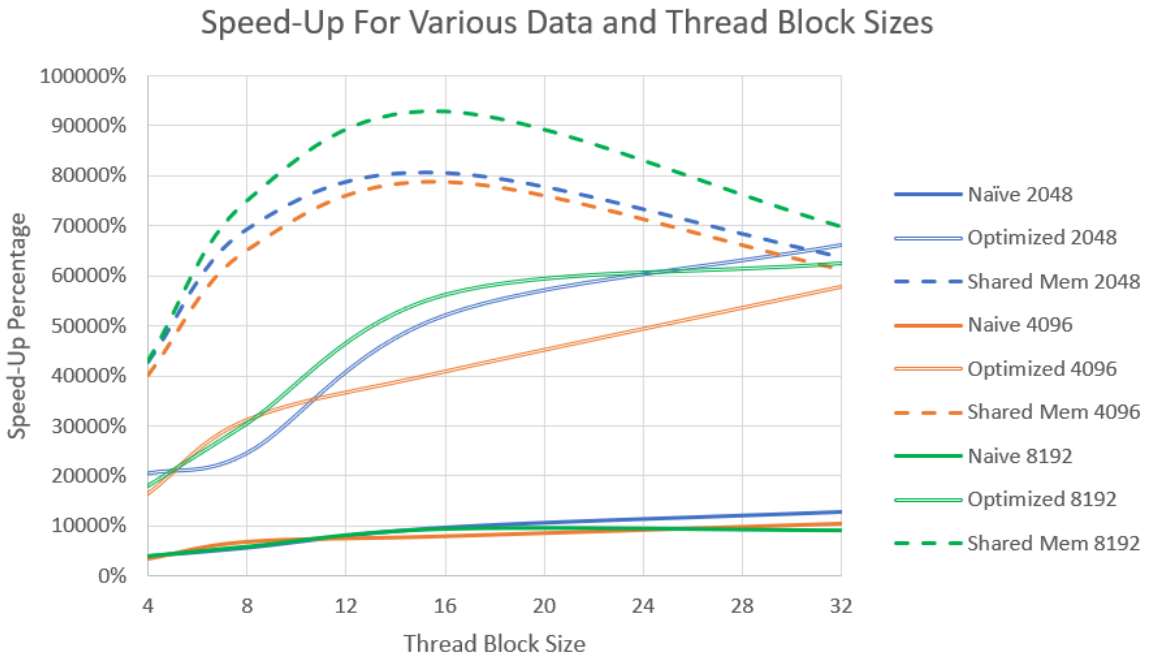

Figure 1: Plot showing the speedup versus the thread block size for various data sizes.

From the plot above it can be seen that the data size was rather minimal in affecting the speed-up of the program, except in some cases. This is perhaps due to the CUDA parallelization being too good at working on problems of this scale and that performance would change more drastically if the problem and data size were even larger. Increasing the thread block size seemed to have a noticeable effect on increasing the speed-up of the program execution. The greatest change though is seen on the advantage of using the optimized algorithm with or without shared memory which creates a very large jump in speed-up for the optimized implementations over the naïve implementation. The speedups seem to converge for the shared memory optimized implementation and the global memory optimized implementation as the thread block size reaches larger values.