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

Parallelization

The code was parallelized by using copying matrix information to each device and subsequently allowing each device to operate on the elements of the matrix given the information for each thread ID in the device. Both the naïve implementation and the optimal implementation were able to be completed using the shared memory space. The difference in implementation comes with converting the matrices to the much simpler (computationally speaking) column major form which allowed for much more coalescing when accessing the shared memory space.

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. For these tests the 1D size and 2D size were decreased by half each time so plots show plots with the x-axis reflecting the 1D size because it is larger and empirically looked better than plotting with respect to the 2D size.

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

	1D Block Size:	128	256	512	1024
	2D Block Size:	4	8	16	32
Data Size	Type				
512	Gold	2.551761	3.52698	3.524081	3.540993
	Naïve	0.511717	0.7593	1.292452	1.322357
	Optimized	0.432436	0.470884	0.726931	0.753656
1024	Gold	28.37228	28.549568	28.229433	28.41391
	Naïve	1.579115	1.166768	3.829957	10.381916
	Optimized	1.847469	1.980952	2.624115	4.888763
2048	Gold	243.674438	238.596573	239.581741	235.480927
	Naïve	5.067243	2.730952	18.498175	34.017876
	Optimized	4.632568	3.859698	9.891169	18.662153

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

	1D Block Size:	128	256	512	1024
	2D Block Size:	4	8	16	32
Data Size	Type				
512	Naïve	499%	465%	273%	268%
	Optimized	590%	749%	485%	470%
1024	Naïve	1797%	2447%	737%	274%
	Optimized	1536%	1441%	1076%	581%
2048	Naïve	4809%	8737%	1295%	692%
	Optimized	5260%	6182%	2422%	1262%

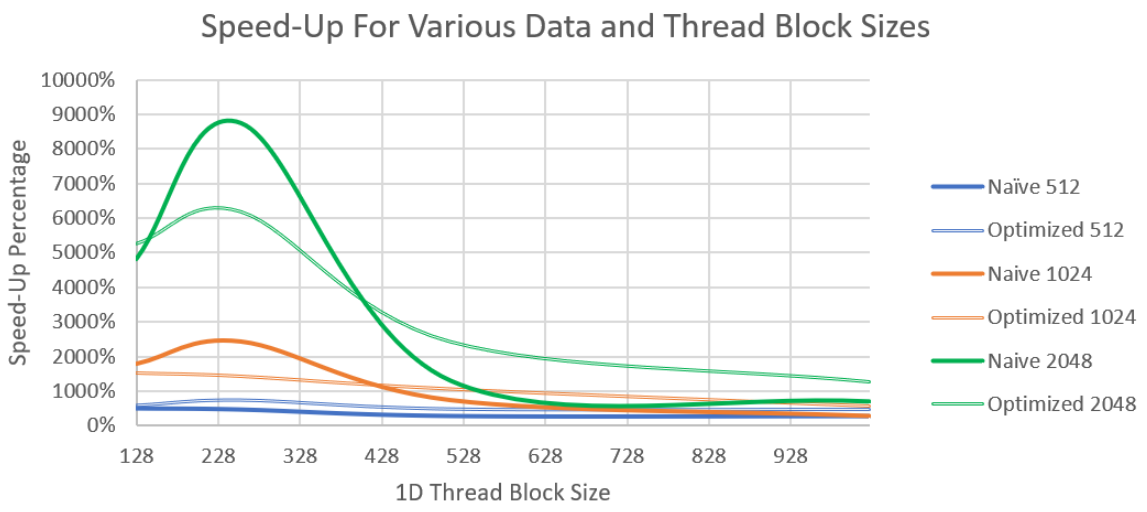

Figure 1: Plot showing the speedup versus the thread block size for various data sizes.

From the plot above it can be seen that as per usual, greater data sizes tend to have a much greater difference on execution time. Another thing to note is that as the thread block size decreases the speed becomes much greater and so does the speed-up. That is for greater thread block sizes, execution time increases. It is also worth noting that the sensitivity of CUDA for thread block sizes greater than 1024 for 1D did not function since there is an upper limit to how large the block size may be for our implementation.