Cameron Calv and Nicholas Sica
ECEC 413 Intro to Parallel Computer Architecture
Assignment 3: Iterative Jacobi Solver (OpenMP)

Parallelization

The code was parallelized by splitting the operations for rows and columns between threads much like what was done in the Pthreads assignment, but this time with OpenMP. Row operations were done in a chunking fashion split between all threads while column operations were done in a striding manner.

Evaluating Parallelization

Below are tables outlining the runtimes of the program run for parallelization. Following the tables are a plot showing the difference of the execution times for all data sizes.

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

	Number of Threads - Jacobi						
Data Size	1	2	4	8	16	32	64
512x512	4.35172	2.39332	1.55134	0.93824	0.72162	4.28039	9.59897
1024x1024	41.83952	19.04787	10.56012	5.85119	14.79794	12.26815	31.76647
2048x2048	355.13611	156.27339	83.19495	45.79100	28.54279	43.68668	75.50692

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

	Number of Threads - Jacobi					
Data Size	2	4	8	16	32	64
512x512	45.00%	64.35%	78.44%	83.42%	1.64%	-120.58%
1024x1024	54.47%	74.76%	86.02%	64.63%	70.68%	24.08%
2048x2048	56.00%	76.57%	87.11%	91.96%	87.70%	78.74%

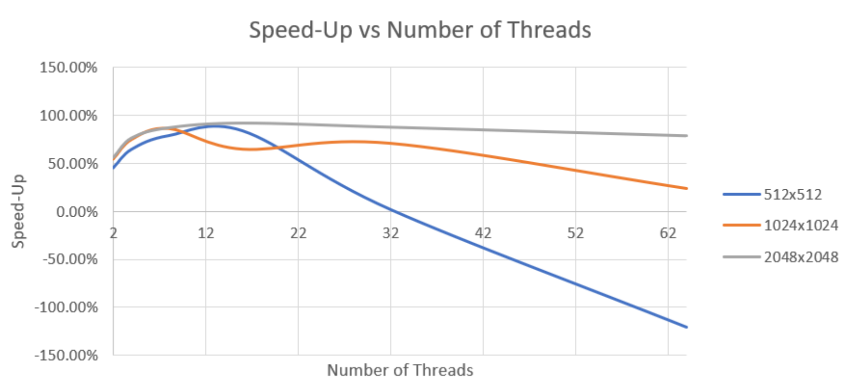

Figure 1: Plot showing the speedup per number of threads for all sizes.

It appears that speedup seem to decrease with the number of threads with the greatest speed up occurring somewhere around 16 threads for the greatest data size. The increasing rate of slowdown seems to come about because of cache sharing and a great number of context switches and a limiting of core number.