Cameron Calv and Nicholas Sica
ECEC 413 Intro to Parallel Computer Architecture
Assignment 3: Gaussian Elimination (OpenMP)

Parallelization

The method by which the code was parallelized follows that the division and elimination steps of the Gaussian elimination algorithm were split amongst available threads much like what was done in the Pthreads assignment, but this time in OpenMP. Upon the completion of generating the random or zeroed matrix of the desired size, the available threads would first take turns choosing which elements in a particular row to divide through. In this stage, each thread used its thread ID to determine which column to start at and then incremented its counter by the total number of threads. This caused each thread to divide its corresponding element of the row in a striding fashion. The elimination step occurred using the chunking method where each thread was told which row to begin elimination of and then to do it for each subsequent row until it reached the starting row of another thread. In this fashion, the full elimination algorithm was split for a requested number of threads.

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 plotted for each data size.

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

	Number of Threads - Gaussian						
Data Size	1	2	4	8	16	32	64
512x512	0.03	0.04	0.02	0.02	0.02	0.23	0.46
1024x1024	0.27	0.18	0.11	0.07	0.06	0.42	1.05
2048x2048	3.8	1.69	0.78	0.53	0.35	1.63	2.72
4096x4096	33.59	20.83	12.66	11.76	19.5	12.41	16.06

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

	Number of Threads - Gaussian					
Data Size	2	4	8	16	32	64
512x512	-33.33%	33.33%	33.33%	33.33%	-666.67%	-1433.33%
1024x1024	33.33%	59.26%	74.07%	77.78%	-55.56%	-288.89%
2048x2048	55.53%	79.47%	86.05%	90.79%	57.11%	28.42%
4096x4096	37.99%	62.31%	64.99%	41.95%	63.05%	52.19%

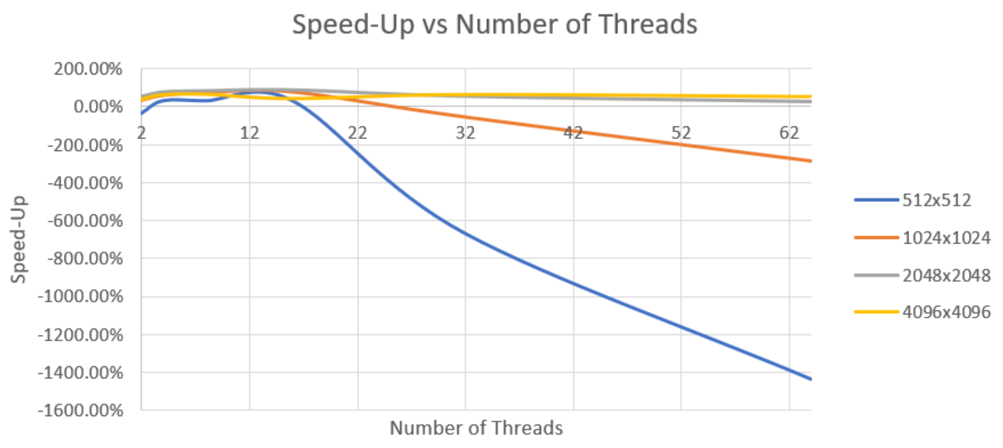

Figure 1: Plot showing the speedup per number of threads for all sizes.

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. Naturally, it is the program runs with the greatest data size that has better improvement with threads but with small numbers of data a more serial option is quicker.