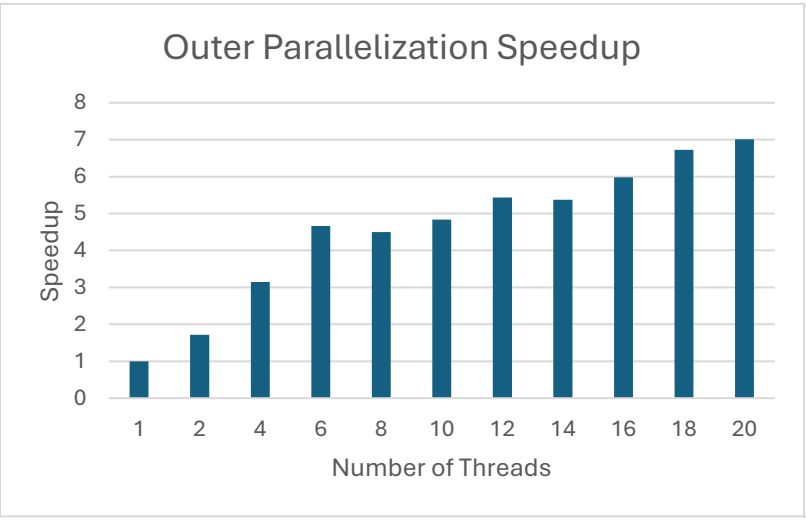
ENSC 453
Lab 1
Nicholas Lagasse
301451354

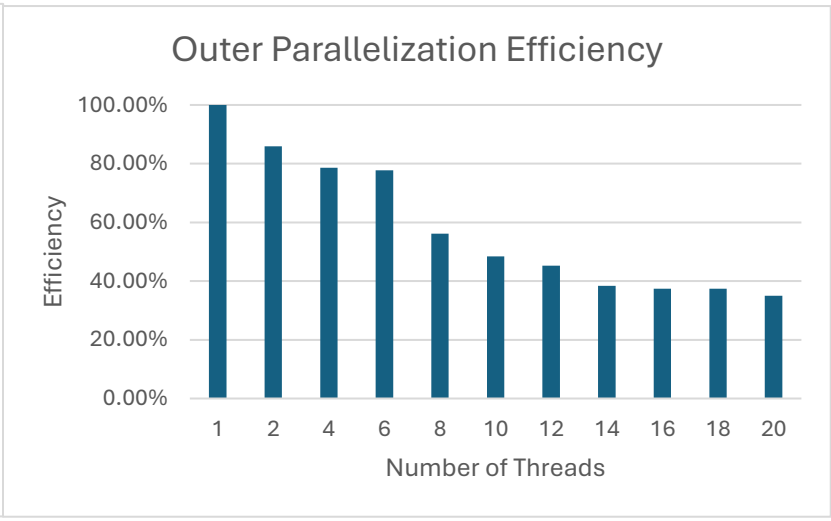
Sequential execution time: 16.599794383
sum of C array = 3212133376.000000

Speedup calculation: sequential_execution_time / parallel_execution_time
Efficiency calculation: speedup / #_cores

1. **Parallelization at the i Loop**

Number of Cores	Parallel Execution Time	Speedup	Efficiency
1	16.59979438	1	1
2	9.656873636	1.718961541	0.85948077
4	5.278957285	3.144521444	0.786130361
6	3.559406668	4.663640862	0.777273477
8	3.692492797	4.495552272	0.561944034
10	3.430206992	4.839298159	0.483929816
12	3.05492746	5.433776939	0.452814745
14	3.088365476	5.374944938	0.383924638
16	2.777201076	5.977166913	0.373572932
18	2.467833619	6.726464157	0.373692453
20	2.367826463	7.010562067	0.350528103



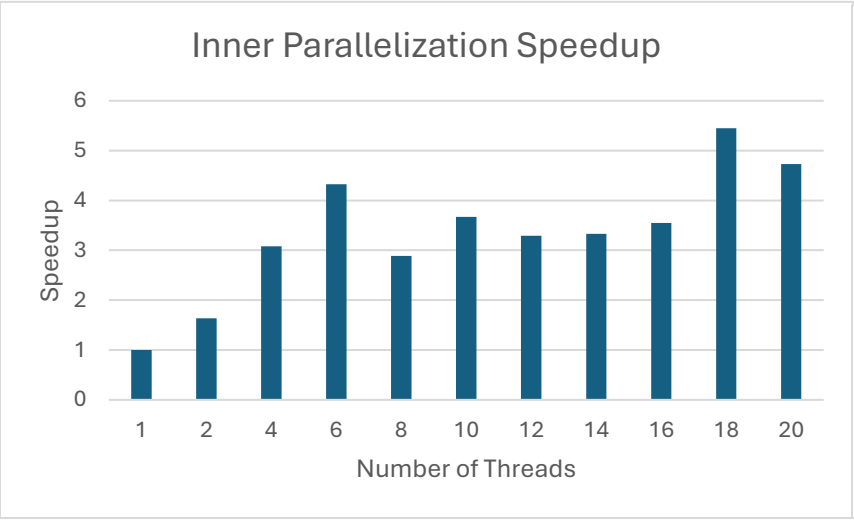


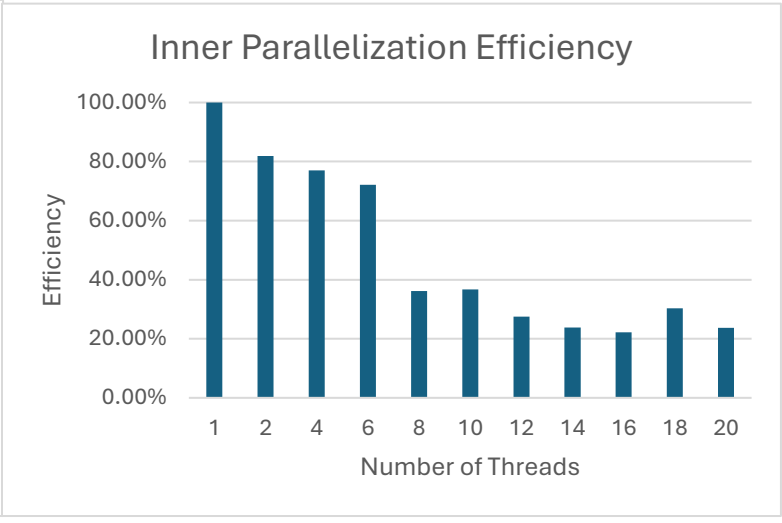
As the number of threads increases, the speedup increases, signalling that parallel computation increases the speed at which the entire computation takes place. The

efficiency also drops off as the number of threads increases, signalling that I'm losing a bit of overall computation power per the number of threads used. There is a slight bump at 6 threads on both the speedup and efficiency charts, which means that I'm getting the best performance per number of threads used at 6 threads.

2. Parallelization at the two j loops

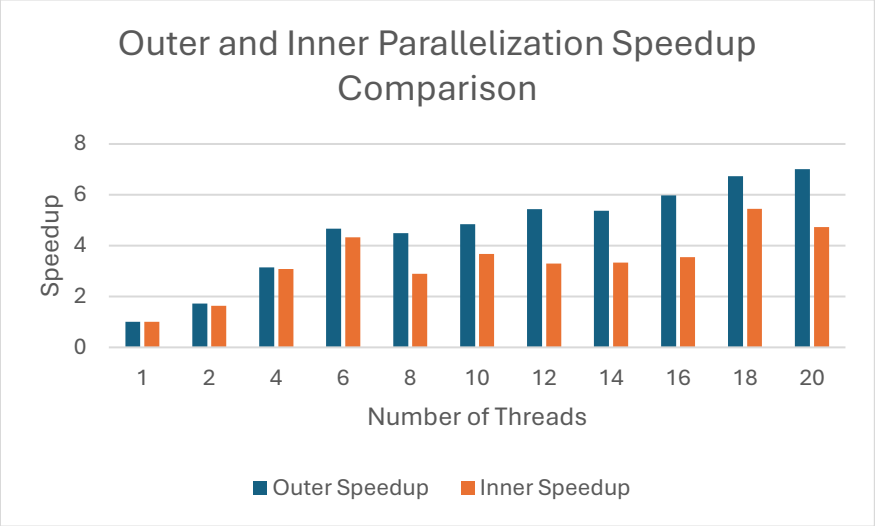
Number of Cores	Parallel Execution Time	Speedup	Efficiency
1	16.59979438	1	1
2	10.13864468	1.63727943	0.818639715
4	5.391667531	3.078786718	0.76969668
6	3.835841584	4.327549514	0.721258252
8	5.74495737	2.889454754	0.361181844
10	4.518110684	3.674056601	0.36740566
12	5.03926876	3.294087927	0.274507327
14	4.983159746	3.331178455	0.237941318
16	4.673489363	3.551905887	0.221994118
18	3.045159261	5.451207297	0.30284485
20	3.509183276	4.730386839	0.236519342





As the number of threads increases, there is a general upwards trend in speedup, as well as a general downward trend in efficiency. By parallelizing the two inner threads, there seems to be a better computation performance as the number of threads increases. Again there is a slight bump in both the 6 and 18 speedup and efficiency, signalling that these numbers of threads tend to perform the best relative to how many threads that were used.

3. Speedup and Parallelization efficiency comparison



As the number of threads increases, there is a general increase in speedup for both the inner loop and outer loop paralellization. However, the outer loop speedup seems to outperform the inner loop speedup as the number of threads increases, which may be due to the calculation being small enough that introducing more threads may overcomplicate the computation required. Both speedup graphs tend to increase as the number of threads increases, and both efficiency graphs tend to decrease as the number of threads increases. However, there is a spike in performance at 6 threads for both inner and outer parallelizations, indiciating a sweet spot in performance per thread no matter the parallelization type. This may also be due to the nature of the calculation not requiring much computational power, and is thus maximized at around 6 threads despite the implementation.