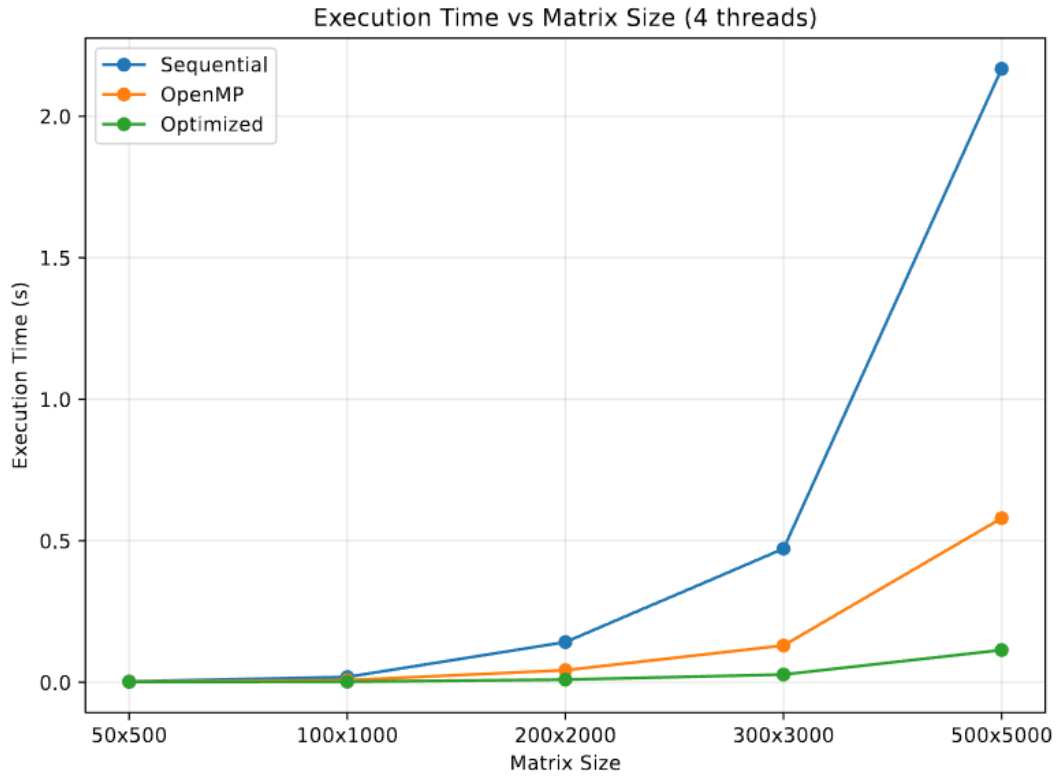
Scaling with Matrix Size

Matrix Size	Sequential (s)	OpenMP (s)	OMP Speedup	Optimized (s)	Optimized Speedup
50 x 500	0.002932	0.001956	1.78x	0.000978	3.00x
100 x 1000	0.018680	0.007414	2.60x	0.001962	9.54x
200 x 2000	0.141970	0.042629	3.38x	0.009395	14.91x
300 x 3000	0.471883	0.129991	3.64x	0.027390	17.25x
500 x 5000	2.167604	0.579802	3.74x	0.113882	19.02x

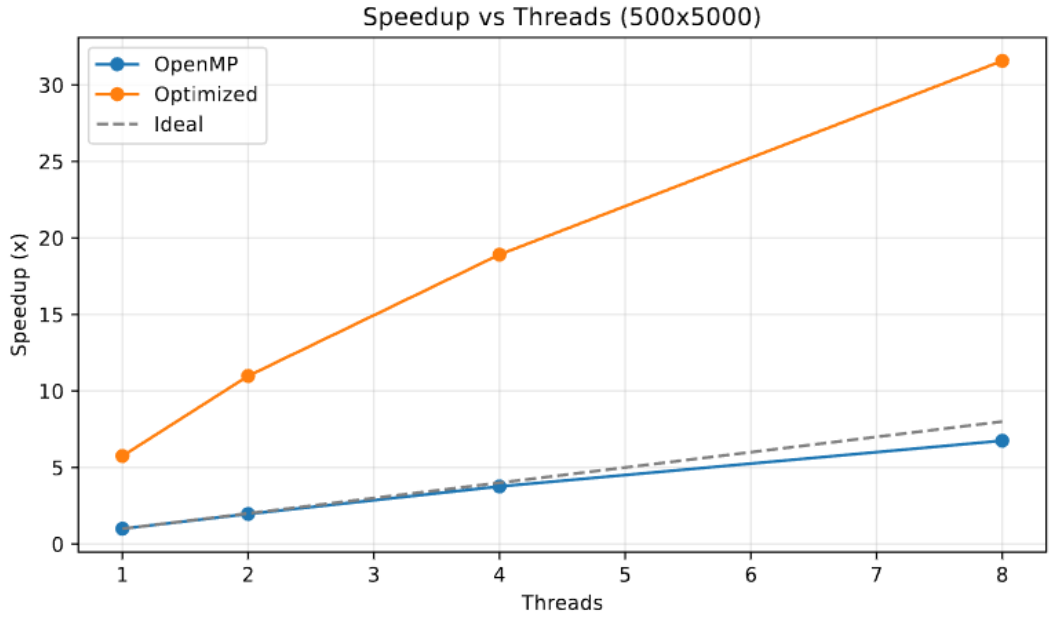
From the results above, execution time increases rapidly as the matrix size grows. This is expected because the correlation algorithm has a high computational complexity. The OpenMP version shows consistent improvement as the workload becomes larger, while the optimized implementation delivers significantly higher speedups due to both parallelism and SIMD vectorization.



Thread Scaling (500 x 5000 Matrix)

Threads	OMP Time (s)	OMP Speedup	OMP Efficiency	Optimized Time (s)	Optimized Speedup	Optimized Efficiency
1	2.162802	1.00x	100.2%	0.377017	5.75x	575.1%
2	1.101517	1.96x	98.1%	0.196956	10.98x	549.1%
4	0.574686	3.76x	94.1%	0.114396	18.91x	472.8%
8	0.320788	6.75x	84.3%	0.068636	31.57x	394.6%

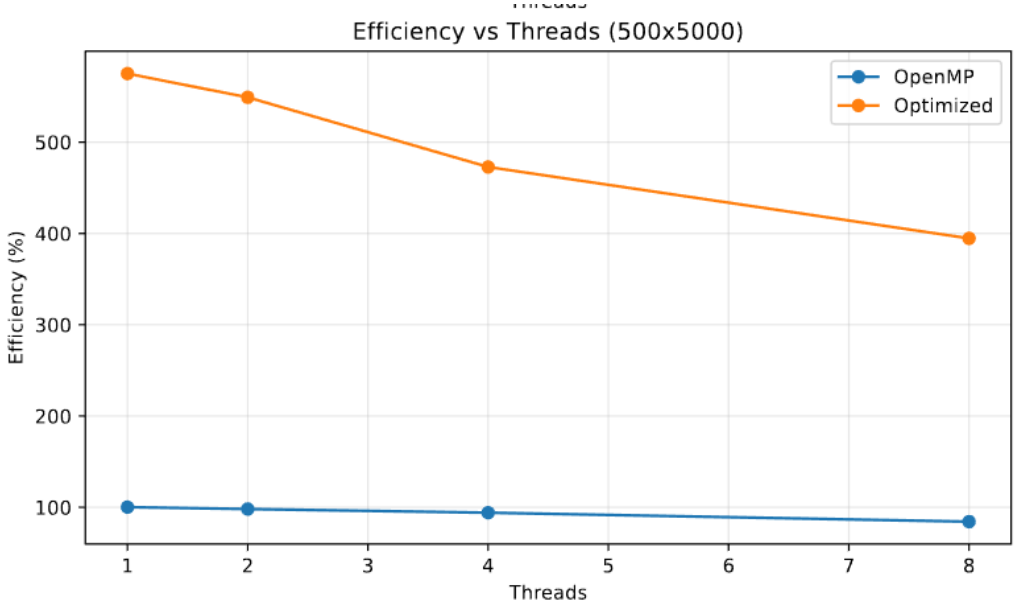
The OpenMP implementation scales well as the number of threads increases. At 8 threads, the speedup reaches 6.75× with around 84% efficiency, which indicates good parallel utilization of CPU cores. As expected, efficiency gradually decreases with more threads due to OpenMP overhead, synchronization costs, and memory bandwidth limitations.



Version Comparison (200 x 2000, 8 Threads)

Sequential Time: 0.142635 s (140,919 correlations/sec)
OpenMP Time: 0.026470 s (759,350 correlations/sec)
Optimized Time: 0.009364 s (2,146,518 correlations/sec)

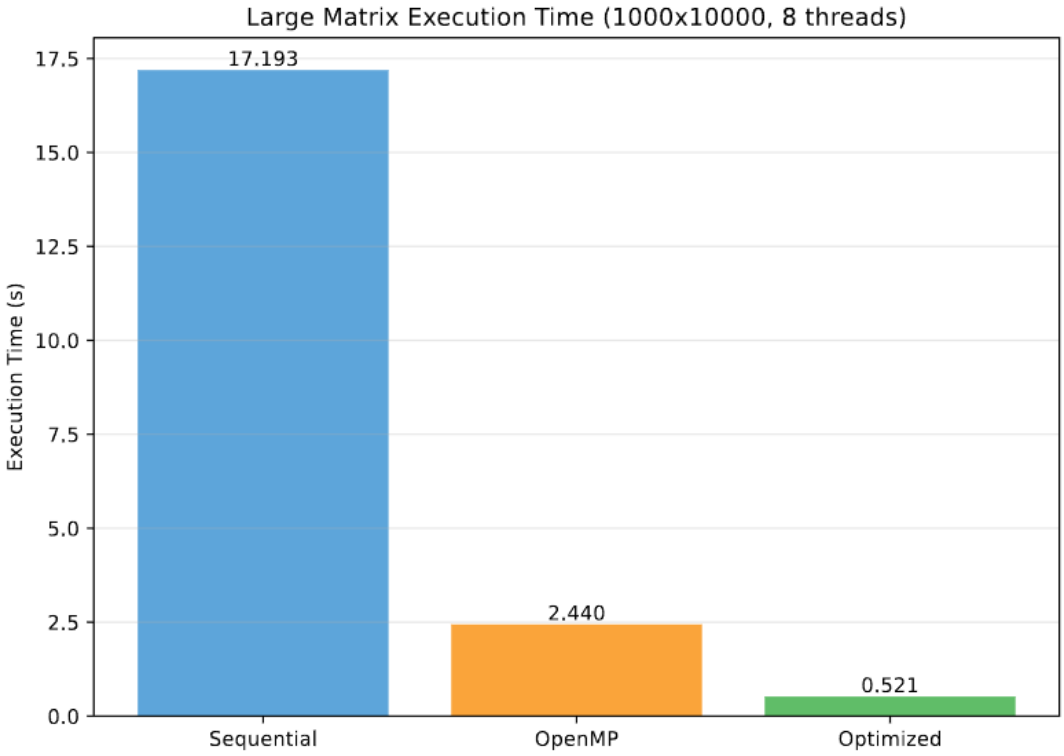
The optimized implementation achieves the highest throughput, clearly demonstrating the combined benefits of multithreading and SIMD vector instructions.



Large Matrix Stress Test (1000 x 10000)

Sequential: 17.192799 s
OpenMP: 2.440407 s (7.06× speedup)
Optimized: 0.520865 s (33.05× speedup)

For large matrices, the benefits of optimization become even more noticeable. The optimized version reaches over 33× speedup compared to the sequential baseline, showing strong scalability and efficient hardware utilization.



Final Conclusion

Overall, the results show that parallelization using OpenMP significantly improves performance, especially for larger matrices. The optimized version provides the best results by combining thread-level parallelism with SIMD vectorization. Although efficiency naturally decreases as the thread count increases, the implementation continues to scale well without reaching performance saturation. This confirms that the algorithm was implemented efficiently and makes good use of the available CPU resources.