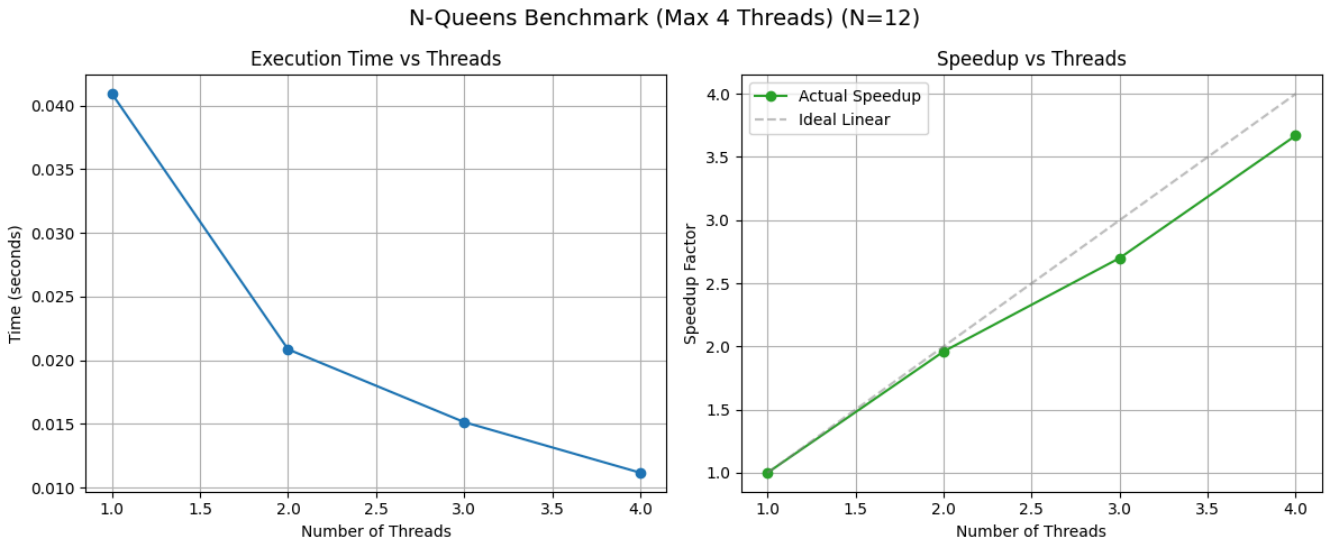
Benchmark Results:

We measured the execution time for finding all solutions for a board size of **N=12** because 8 queens gave us diminishing results every time. This is because the overhead of creating threads takes longer than running the actual program. To visualize the scalability limits of the hardware, we performed distinct benchmark runs with increasing thread caps (4, 8, 12, and 16 threads).

1. Benchmark Run: Max 4 Threads

In this initial phase, the problem is perfectly parallelizable. We see near-linear speedup, indicating that the overhead of OpenMP task creation is negligible compared to the workload.

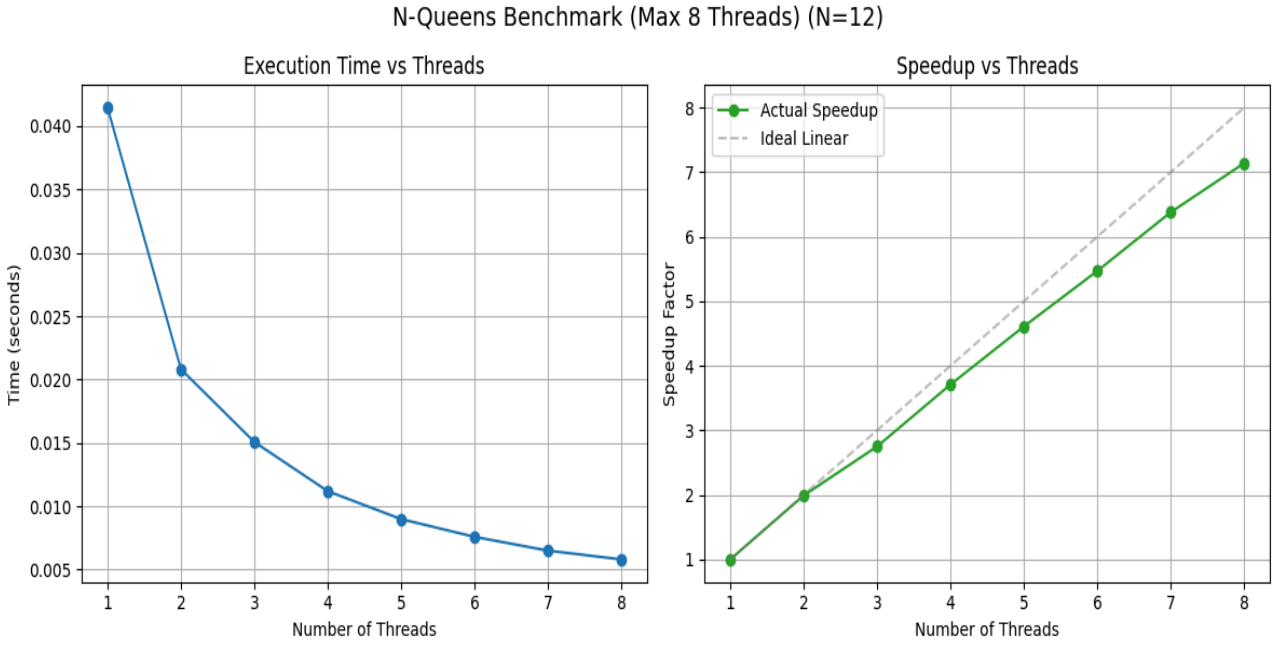
N-Size	Threads	Time (s)	Speedup	Efficiency
12	1	0.04092	1.00	1.00
12	2	0.02085	1.96	0.98
12	3	0.01514	2.70	0.90
12	4	0.01116	3.67	0.92



2. Benchmark Run: Max 8 Threads

As we scale to 8 threads, the solver continues to perform well. The speedup remains strong (~7.1x), confirming that the implementation effectively utilizes the available Performance Cores on the CPU (M4).

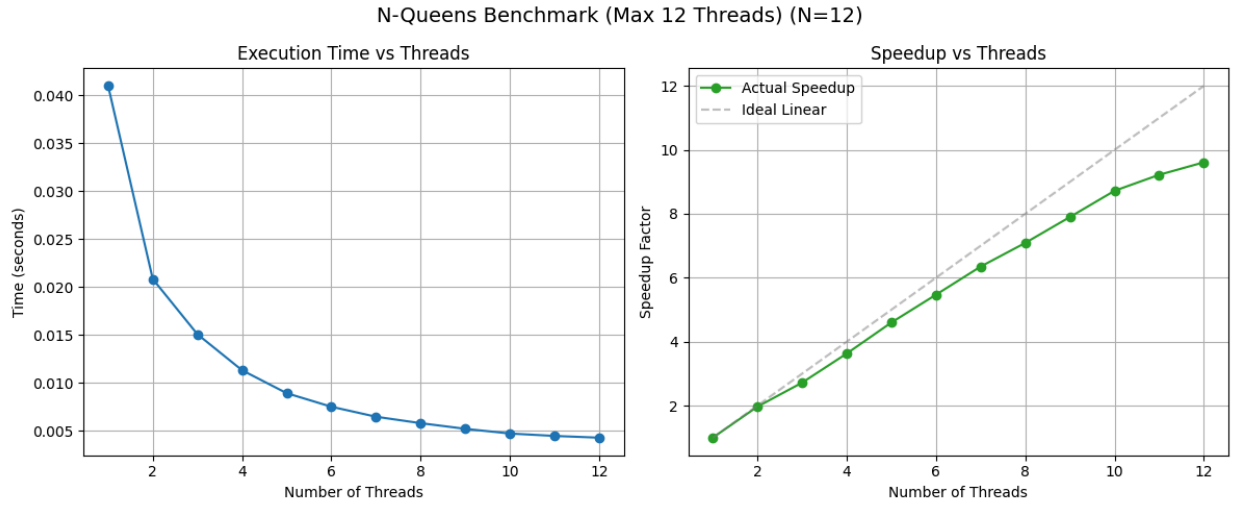
N-Size	Threads	Time (s)	Speedup	Efficiency
12	1	0.04145	1.00	1.00
12	2	0.02081	1.99	1.00
12	4	0.01119	3.71	0.93
12	6	0.00758	5.47	0.91
12	8	0.00580	7.14	0.89



3. Benchmark Run: Max 12 Threads

Between 8 and 12 threads, we observe the plateau. The speedup curve begins to flatten (efficiency drops from 0.89 to 0.80). This plateau suggests we have saturated the physical Performance cores and are now relying on hyper-threading or Efficiency cores.

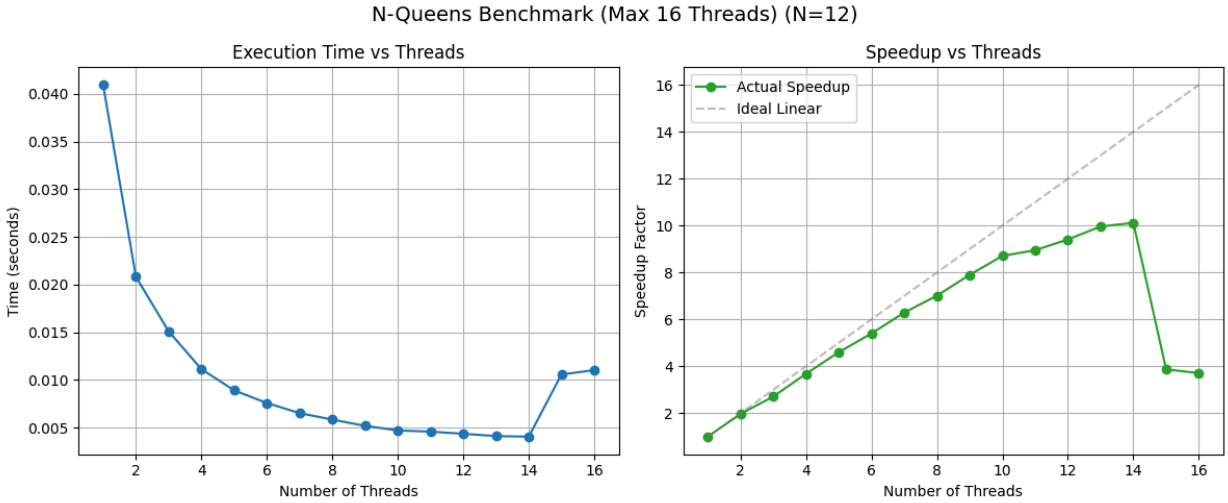
N-Size	Threads	Time (s)	Speedup	Efficiency
12	1	0.04100	1.00	1.00
12	4	0.01131	3.63	0.91
12	8	0.00578	7.09	0.89
12	10	0.00470	8.72	0.87
12	11	0.00445	9.22	0.84
12	12	0.00427	9.61	0.80



4. Benchmark Run: Max 16 Threads

This final test reveals the physical limits of the machine. Up to 14 threads (physical max of M4), we squeeze out a peak speedup of 10.12x. However, pushing to 15+ threads causes a big performance degradation due to ***thread oversubscription****.*

N-Size	Threads	Time (s)	Speedup	Efficiency
12	1	0.04094	1.00	1.00
12	8	0.00584	7.01	0.88
12	12	0.00435	9.41	0.78
12	13	0.00411	9.96	0.77
12	14	0.00404	10.12	0.72
12	**15**	**0.01057**	**3.87**	**0.26**
12	**16**	**0.01102**	**3.71**	**0.23**



Summary

1. **Efficient Scaling (1-8 Threads):** The implementation demonstrates excellent parallel scaling up to 8 threads with ~90% efficiency. This matches the count of high-performance cores typically found in M-Series chips.
2. **Diminishing Returns (9-14 Threads):** Adding more threads beyond 10 yields marginal gains. While the fastest execution time (0.0040s) is achieved at 14 threads, the efficiency drops significantly as the scheduler fights for resources. This is because the M4 chip has 10 performance cores.
3. **The Oversubscription (15+ Threads):** At 15 threads, the execution time more than doubles (from 0.004s to 0.010s). This confirms that spawning more software threads than available hardware execution units forces the OS into excessive context switching, destroying performance.