

# Parallel Solution to the N-Queens Problem (HW2)

Marios Stavrou, Panayiotis Charalambous

January 2025

## Performance Evaluation

We evaluate the performance of a backtracking solver for the  $N$ -Queens problem implemented in three variants:

- Sequential implementation
- OpenMP-based parallel implementation
- Pthreads-based parallel implementation

Execution time is measured using wall-clock timing, and only the parallel computation region is timed. Each configuration is executed 10 times, and the mean execution time is reported to reduce noise due to system variability.

## Experimental Setup

Parallelism is introduced by distributing the search over the first column placements. Each thread explores a subtree independently using a private board. The OpenMP implementation uses:

- `#pragma omp parallel for`
- Reduction on the global solution counter

The number of threads is controlled according to the number of queens to be placed.  $N$  queens implies  $N$  threads. This works for  $N$ s up to 20 but then scaling takes significant computation time.

## Results

$N = 10$

Implementation	Time ( $\mu$ s)	Speedup
Sequential	9945	1.00
OpenMP	3590	2.77
Pthreads	2991	3.32

$N = 8$

Implementation	Time ( $\mu$ s)	Speedup
Sequential	506	1.00
OpenMP	664	0.76
Pthreads	212	2.39

## Discussion

For  $N = 10$ , both parallel implementations outperform the sequential version. The OpenMP version achieves a speedup of approximately  $2.8\times$ , while the Pthreads version reaches over  $3.3\times$ . This improvement is due to the increased computational workload, which outperforms thread creation, scheduling, and synchronization overhead.

In contrast, for  $N = 8$ , the problem size is too small to benefit from OpenMP parallelization. The overhead of thread management and dynamic scheduling dominates execution time, resulting in slower performance than the sequential version. This demonstrates that parallelism is not always beneficial for fine-grained or short-running tasks.

The Pthreads implementation performs best in both cases, particularly for  $N = 8$ . This is likely due to lower runtime overhead and finer control over thread management compared to OpenMP, which introduces additional abstraction costs.

## Advantages and Disadvantages

### OpenMP

- **Advantages:**
  - Simple and concise parallelization using compiler directives
  - Automatic workload distribution and reduction handling
  - Good scalability for sufficiently large problem sizes
- **Disadvantages:**
  - Higher runtime overhead for small workloads
  - Performance sensitive to scheduling policy and chunk size

### Pthreads

- **Advantages:**
  - Lower overhead and better performance for small tasks
  - Fine-grained control over threads and synchronization
- **Disadvantages:**
  - More complex and error-prone programming model
  - Reduced portability and readability compared to OpenMP

## Unnecessary Parallelization

In our program, each thread trying to place a new queen on the board, calls a function called **safe**, which checks if the queen can be placed in a specific position without attacking any of the previously placed queens on the board.

This function is implemented using 3 for loops, 1 checking the left upper diagonal, 1 checking the left lower diagonal, and 1 checking the left horizontal positions.

After trying to parallelize this function, we came to the conclusion that since it's already a really simple fast procedure, and since it is called recursively many time, parallelizing it adds extremely huge overhead. This is due to the fact that each time **safe** is called after a recursive call of the main program, 3 threads must be initialized, configured, synchronized and then destroyed.

As a result, the execution time for the 8 queens problem became approx. 2 million microseconds, compared to the 664 microseconds the program needed to execute before parallelizing the **safe** function.

Concluding, we found out that for small procedures, parallelization could be costly.

## Conclusion

Parallelization significantly improves performance for larger instances of the  $N$ -Queens problem. However, for small problem sizes, overhead can negate the benefits of parallel execution.