POLITECNICO
MILANO 1863

Challenge 2

Advanced Algorithms and Parallel Programming

Camilo José Sinning López, Oliver Mosgaard Stege
& Raul Eduardo Vergara Lacouture

November 20, 2024

1 Experimental Setup

The program implements a parallel Sudoku solver using OpenMP. The algorithm recursively explores all possible solutions to a given Sudoku puzzle by assigning numbers to unfilled cells, ensuring no conflicts in rows, columns, or subgrids.

1.1 Hardware and software

All experiments were conducted on a laptop with the following processor details:

- **Model:** AMD Ryzen 5 7520U with Radeon Graphics
- **Base Clock:** 2.8 GHz
- **Max Boost Clock:** Up to 4.3 GHz
- **Cores:** 4
- **Threads:** 8

1.2 Algorithm parameters

- Grid Size: Dimension of the Sudoku puzzle (e.g., 9x9).
- Block Size: Size of subgrids (e.g., 3x3 for standard Sudoku).
- Max Depth: Specifies the depth for task creation in the parallel recursion.
- Threads: Maximum number of threads to use in the parallel algorithm.
- sudoku: The Sudoku puzzle to solve.

1.3 Executions

1.3.1 Fixed depth

The first set of experiments was conducted with a fixed max_depth of 3. For each thread count, 3 samples were collected for 9x9 and 25x25 Sudoku puzzles.

1.3.2 Variable depth

The second set of experiments was conducted with a variable max_depth. For thread counts ranging from 1 to 12, experiments were conducted for each max_depth value in 3, 5, 7, 9. Three samples were collected for each configuration.

2 Performance Measurements

Summarized results are saved in a CSV file with columns:

1. Execution time.
2. Grid size.
3. Block size.
4. Number of threads.

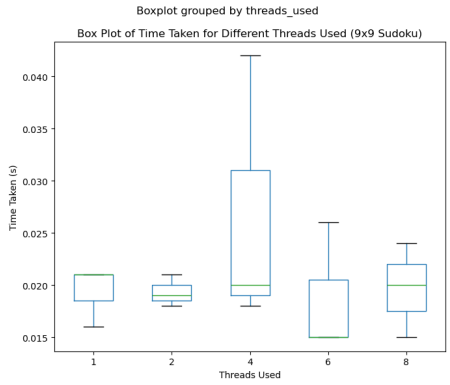
5. Number of solutions.
6. Max depth.

Sample output: The program outputs the timing and results to the console and appends them to the times.csv file.

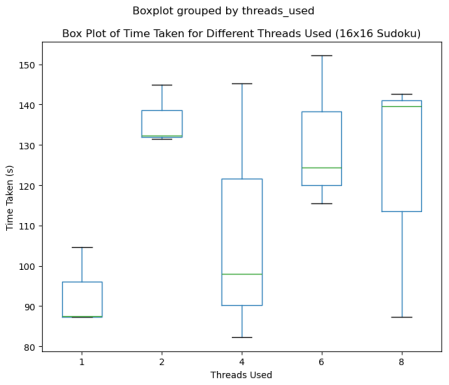
3 Results

The following results were taken from times.csv file, and only the threads available in the machine were used for to create the plots since more threads would not be beneficial.

3.1 Fixed depth



(a) Execution time for 9x9 Sudoku puzzles with fixed depth.

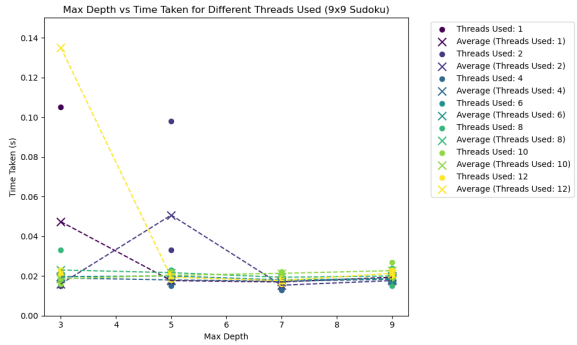


(b) Execution time for 16x16 Sudoku puzzles with fixed depth.

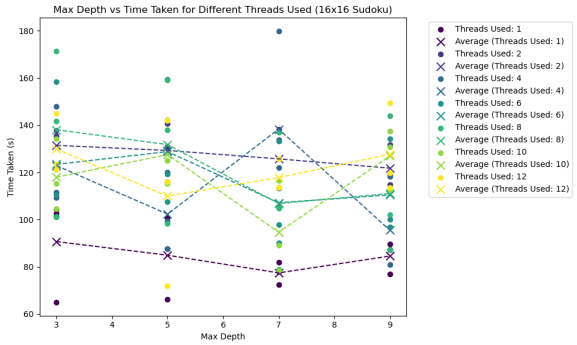
Figure 1: Comparison of execution times for fixed depth Sudoku puzzles.

In the fixed-depth experiments, the results indicate that increasing the number of threads does not consistently lead to significant performance improvements, especially for the 16x16 Sudoku puzzles. While some reduction in execution time is observed for lower thread counts, the variance increases with more threads, and the performance gains diminish. This suggests limited parallelization efficiency due to factors like overhead from thread management, synchronization, or insufficient workload distribution across threads at fixed depths.

3.2 Variable depth



(a) Execution time for 9x9 Sudoku puzzles with variable depth.



(b) Execution time for 16x16 Sudoku puzzles with variable depth.

Figure 2: Comparison of execution times for variable depth Sudoku puzzles.

For variable-depth experiments, performance trends are similarly unclear. The 9x9 Sudoku puzzles show minor gains with increasing thread counts at specific depths but reveal diminishing returns at higher depths. The 16x16 puzzles demonstrate significant variability, with some thread configurations performing worse as depth increases. These results highlight challenges in balancing task granularity and overhead when managing parallel recursion, emphasizing the need for better load-balancing strategies to maximize thread utilization effectively.

4 Design Choices

1. **Parallelization with OpenMP:** The algorithm uses OpenMP to parallelize the Sudoku solving process. More specifically using the tasks directive.
2. **Recursive Backtracking:** The core of the algorithm is a recursive backtracking approach.
3. **Depth Limitation:** To control the granularity of parallel tasks and avoid excessive task creation, the algorithm limits the depth of parallel recursion using the depth and max_depth parameters.
4. **Board Copying:** To avoid shared state issues between parallel tasks, the algorithm creates a copy of the Sudoku board for each task.
5. **CSV Logging:** The algorithm logs the performance data to a CSV file.

5 Conclusions

The results of the experiments suggest it is not clear that increasing the number of threads consistently leads to performance improvements in the parallel Sudoku solver using a backtracking algorithm. The fixed-depth experiments show limited gains with more threads, while the variable-depth experiments exhibit significant variability in performance. The challenges in balancing task granularity, load distribution, and overhead management highlight the need for more sophisticated parallelization strategies to maximize thread utilization effectively.