POLITECNICO
MILANO 1863

Challenge 2

Advanced Algorithms and Parallel Programming

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

November 21, 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

Since the processor has 8 threads, but the experiments were conducted with thread counts ranging from 1 to 12, the results also include the execution time for thread counts higher than the number of available threads to assess the impact of the overhead of creating additional threads.

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 16x16 Sudoku puzzles. On the other hand, the 25x25 Sudoku puzzle was only tested once per thread count due to the high execution time.

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 9x9 and 16x16 Sudoku.

2 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.

2.1 Fixed depth

In the fixed-depth experiments, the results indicate that overall increasing the number of threads lead to performance improvements in the parallel Sudoku solver. The 16x16 Sudoku puzzles (Figure 2) show consistent reductions in execution time as the number of threads increases. The 25x25 Sudoku puzzle (Figure 3) also demonstrates similar trends, although in 4 threads the execution time is higher than in 2 threads. The 9x9 Sudoku puzzles (Figure 1) exhibit more variability, with some thread configurations performing worse than others. These results suggest that the performance gains from parallelization are more pronounced in larger Sudoku puzzles, where the overhead of task creation is amortized over more work.

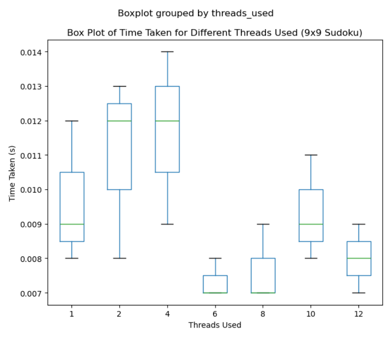

Figure 1: Execution time for 9x9 Sudoku puzzles with fixed depth.

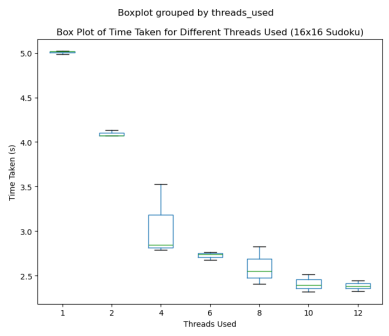

Figure 2: Execution time for 16x16 Sudoku puzzles with fixed depth.

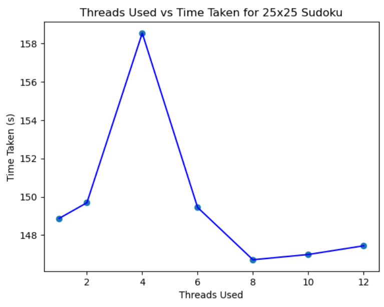

Figure 3: Execution time for 25x25 Sudoku puzzles with fixed depth.

2.2 Variable depth

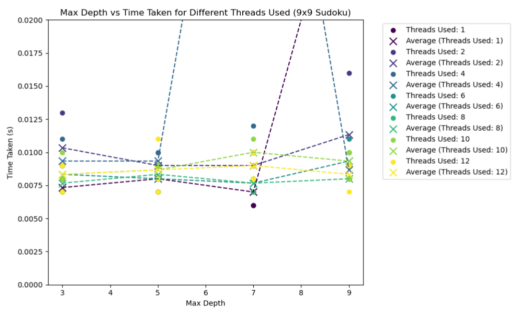

Figure 4: Execution time for 9x9 Sudoku puzzles with variable depth.

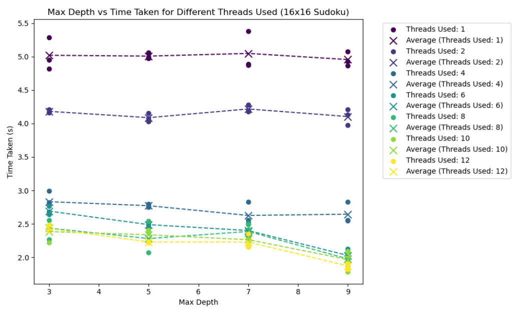

Figure 5: Execution time for 16x16 Sudoku puzzles with variable depth.

For variable-depth experiments, performance trends are similarly unclear for the 9x9 Sudoku puzzle (Figure 4), while the 16x16 Sudoku puzzle (Figure 5) shows more consistent improvements with more threads. Although none of them seemed to have a clear trend with respect to the depth of the recursion.

3 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.

4 Conclusions

In summary, the parallel Sudoku solver demonstrates performance improvements with increasing thread counts, especially for larger Sudoku puzzles. The fixed-depth experiments show consistent reductions in execution time as the number of threads increases, while the variable-depth experiments exhibits small changes in performance. The results suggest that the parallel algorithm is effective in reducing the execution time of Sudoku puzzles, particularly for larger grid sizes.