**Parallel Backtracking Method:** Classic backtracking approach

1. Identify an empty cell
2. Try each digit (1-9) that can legally be placed
3. For each valid digit, recursively solve the resulting board.
4. Continue until all cells are filled or no valid moves remain

**OpenMP Tasks in Practice**

- After entering a parallel region, I used a #pragma omp single to start backtracking in just one thread
- Inside the backtracking function, each valid digit triggers a new task. This allows multiple branches of the solution space to be processed simultaneously on multi-core CPUs.
- Taskwait ensures that the parent function waits for all sub-branches to complete at the current level before moving on

**Shared State and Synchronization**

- I used a global flag (solutionFound) to indicate when a valid solution was discovered. Once it is set, other tasks can return early, saving time.
- To prevent data races when setting the solution, I used a critical section to ensure that only one task writes the final solution at a time.

**Local Board Copies**

- Each parallel branch needs its own copy of the board. This avoids the complexity of locking. Instead, each task effectively explores its own partial solution without overwriting another's changes.

**Benefits and Overheads**

- Benefit: When the puzzle is difficult or the search space is large, parallel tasks significantly reduce total solve time by distributing work
- Overhead: The cost of task creation for smaller puzzles can offset the gain.