Parallelization in Sudoku Solving Algorithm

Team Members: Bella Liu, Derek Duenas

URL: https://bella713.github.io/15418-final-project/

Summary

The project aims to implement a Sudoku solver using backtracking algorithm, and optimize it with parallelization using OpenMP and MPI.

Background

Sudoku is a classical logic-based combinatorial puzzle game originated from Japan. In classic Sudoku, the player is given a partially filled 9 x 9 grid. The objective is to fill the grid with digits such that each row, column, and 3 x 3 sub-grid contain all the digits from 1-9. In this project, different N * N of Sudoku boards will be tested, such as 16 x 16 and 25 x 25, which will better assess the performance of the project.

There are multiple algorithms to solve Sudoku.

1) Brute-force algorithm: generate all possible configurations by filling all the empty cells. Try every configuration one by one until the correct configuration is found.
2) Backtracking: Before assigning a number, check whether it is safe to assign. Check that the same number is not present in the current row, current column and current 3X3 subgrid. After checking for safety, assign the number, and recursively check whether this assignment leads to a solution or not. If the assignment doesn’t lead to a solution, then try the next number for the current empty cell. And if none of the number (1 to 9) leads to a solution, return false and print no solution exists.

Parallelization can be applied when testing different possible answers for each grid. The inherent nature of the problem allows for parallelization to optimize it.

The Challenge

The algorithm utilizes recursive functions, which can be hard to parallelize. The backtracking algorithm utilizes depth-first search, which depends on an implicit stack structure. To implement parallization, the algorithm needs to be modified to be parallelization-friendly. For example, we can combine DFS with BFS, since BFS is more intuitive to parallelize. The algorithm can find all possible boards using BFS first, then search through each possibility using DFS with parallelization.

In addition, the time complexity of the backtracking algorithm is $O(K^{(N*N)})$, where k is the board size (9, 16, 25, etc.). The computation time increases exponentially with the increase of board size and difficulty of the given board, which adds to the difficulty of the project as well.

Resources

The code will be written from scratch.
Some references:

- Naive implementation of backtracking algorithm: https://www.geeksforgeeks.org/sudoku-backtracking-7/#
- Previous related 15418 project: https://www.andrew.cmu.edu/user/cshan1/15418/proposal/

Goals and Deliverables

- **Plan to achieve:**
    - The code would take in one or multiple boards of size 9 x 9, 16 x 16 or 25 x 25 from an input file, generate answers and output the corresponding solutions if a solution can be found.
    - Parallelization of the sudoku solver using OpenMP
    - Parallelization of the sudoku solver using MPI
    - An analysis report that shows the speedup performance both OpenMP and MPI parallelization of the code on GHC and PSC machines. Related graphs and tables should be constructed to display the results.

- **Hope to achieve:**
    - Add parallelization by CUDA
    - Add visualization for the backtracking algorithm, which can be shown in the poster session*

Platform

The project will be written in C++, and the final performance will be analyzed on GHC and PSC machines. The code language and platforms are chosen because they are familiarized with from previous course assignments, and would be an intuitively good choice for parallelization in this project.