Sudoku Solvers with OpenMP and MPI

Zhongyi Cao, Liyao Fu

Website URL

https://zhongyi-liyao-15418-site.webflow.io/

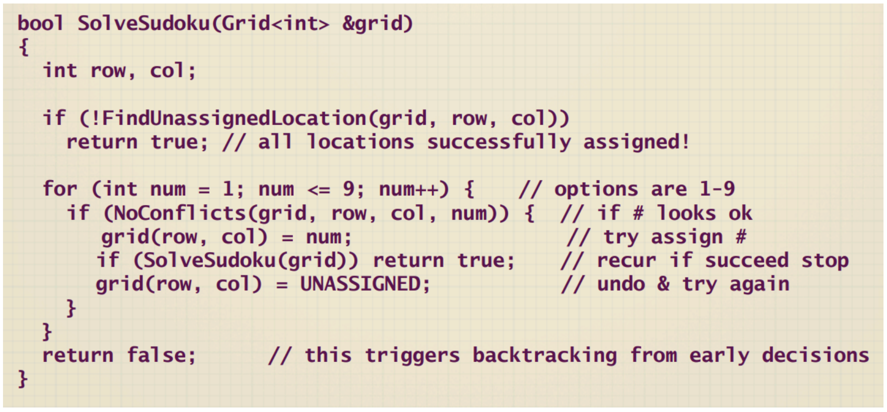
Summary

We plan to come up with algorithms for OpenMP sudoku solver and MPI sudoku solver and implement them. After implementation, we will evaluate and compare the performance of these two algorithms in different sudoku puzzles (sub doku, super doku, and even jigsaw sudoku).

Background

In the Sudoku problem, the objective is to fill a 9 × 9 grid with digits so that each column, each row, and each of the nine 3 × 3 subgrids that compose the grid contain all of the digits from 1 to 9. The goal of a Sudoku solver is to give solutions to an input that has certain predetermined digits so that the answer will fulfill the requirements for all the columns, rows, and subgrids.

While the problem is straightforward, the time complexity can be very large. If each Sudoku square has n choices of numbers, and there are m squares to fill in, a naive Sudoku solver will take O(n^m) to finish, which will be terrible even on the latest CPUs. The Intel Core i9, with 18 cores and a price tag of $1,999, can do roughly a trillion operations per second. However, if there are around 30 empty spaces to fill in, the i9 CPU will take around (9^30)/(10^12)=4.24e+16 seconds to compute the result, which is over one billion years. In fact, it was proved in 2001 that the Sudoku problem is NP-hard.[1]



```
bool SolveSudoku(Grid<int> &grid)
{
  int row, col;

  if (!FindUnassignedLocation(grid, row, col))
    return true; // all locations successfully assigned!

  for (int num = 1; num <= 9; num++) {    // options are 1-9
    if (NoConflicts(grid, row, col, num)) {  // if # looks ok
       grid(row, col) = num;                 // try assign #
       if (SolveSudoku(grid)) return true;   // recur if succeed stop
       grid(row, col) = UNASSIGNED;          // undo & try again
    }
  }
  return false;      // this triggers backtracking from early decisions
}
```

Naive Sudoku Solver Algorithm (Stanford University)

While the naive solution is too time consuming, parallelism can be a very useful tool in improving the efficiency to solve the Sudoku problem. For example, we can parallelize the steps to try different numbers for each grid, and gather the information for whether the assignment is successful from the result of each thread that handles a subgrid, a row, or a column.

The Challenge

The difficult parts to parallelize in the Sudoku problem are that:

1. We need to implement the solver in two strategies: OpenMP and MPI. In particular, for MPI, we have to design how to communicate and update between all processes when our algorithm uses backtracking. In the backtracking search, the result of the subgrids' trials should be sent back to the master thread, and the master thread will determine if the subgrids make a valid combination. While the subgrids can have different numbers of possible combinations, the main thread needs to make sure that each combination has been used to align with another.

2. The workload imbalance in multiple threads will be a major problem when the number of missing spaces varies a lot for each subgrid. For example, for a 9x9 sudoku, a subgrid can have only 1 missing space, while another has 8. At such times, we must deal with work stealing and let idle threads handle some work.

Resources

For the OpenMP version of the solver, we will use the GHC machines for simulation and evaluation. For the MPI version of the solver, we will use the PSC machines.
For code base, we will refer to multiple papers and online resources about the Sudoku solver and parallel problem solving. Specifically, we will be using a sequential version of the Sudoku solver as our starter code, and implement our parallel solver. We compare our results with existing solvers. [3]

Goals and Deliverables

- Goals we plan to achieve:
  We plan to design parallel Sudoku solvers in both OpenMP and MPI, which will have significant performance boost from sequential solvers. Since parallelism allows concurrent computation of different subgrids in the Sudoku problem, we hope to accelerate the computation and reduce the time to a negligible amount.

  We plan to build general Sudoku solvers, which means they can not just solve the 9x9 Sudokus, but also any NxN Sudokus where N is a perfect square number. We will refer to general Sudoku solving techniques when building such Sudoku solvers.

  We plan to evaluate and analyze the performance of the OpenMP and MPI solvers, and determine under what circumstances the different solvers perform the best. We will do so by testing our code on different, complex datasets and evaluating the result for different thread numbers.

- Goals we hope to achieve:
  We hope to design extra Sudoku solvers for other versions of the Sudoku problem, such as the sub doku, super doku, and even jigsaw sudoku. It will require different techniques to solve these problems, but the general parallel methods will still apply.

- Demo:
  We will demonstrate our Sudoku solver at the poster session, and people can test the performance of the Sudoku solver or give the solvers their own Sudoku problem and see how they solve it.

- Analysis:
  We will analyze the performance of the OpenMP and MPI solvers for different Sudoku sizes, and different number of threads. We will conclude which solver performs better under what conditions, and summarize the results.

Platform Choice

We choose to test the OpenMP version on a GHC machine, as it is easy to access with a sufficient number of threads for a general Sudoku problem. We choose to test the MPI version on a PSC machine as it allows the testing of MPI code with many available threads. We choose to use C++ because it supports both OpenMP and MPI and we can rewrite the versions from existing code.

Schedule

Week 1 (3.27 - 4.2)	● Meet with professor to discuss project ideas ● Create a web page for the project ● Write project proposal
Week 2 (4.3 - 4.9)	● Design and implement OpenMP sudoku solver
Week 3 (4.10- 4.16)	● Debug OpenMP sudoku solver ● Design and implement MPI sudoku solver
Week 4 (4.17 - 4.23)	● Debug MPI sudoku solver ● Write milestone report
Week 5 (4.24 - 4.30)	● Analyze the performance of the OpenMP and MPI solvers ● Improve the performance of OpenMP and MPI solvers based on analysis
Week 6 (5.1 - 5.5)	● Write final report ● Prepare for demo and poster session

Reference

1. https://www.sciencedirect.com/science/article/pii/S097286001630038X#:~:text=In%202003%2C%20the%20generalised%20Sudoku,be%20certified%20in%20polynomial%20time.
2. https://see.stanford.edu/materials/icspacs106b/lecture11.pdf
3. https://github.com/huaminghuangtw/Parallel-Sudoku-Solver