Parallelized Sudoku Solver using OpenMP
Report

Saurabh Kumar	Amit Kumar	Ali imam	Amit Kumar
16D100018	16D070034	16D100017	16D100023

1. Introduction

A standard Sudoku puzzle contains 81 grids - 9 rows and 9 columns. 9 non-overlapping blocks, each block consists of 3 rows and 3 columns. From a mathematical perspective, it has been proven that the total number of valid Sudoku puzzles for 9x9 is equal to **6,670,903,752,021,072,936,960** or **$6.671 \times 10^{21.}$** This large number also directly eliminates the possibility of solving the puzzle with brute force technique in a reasonable amount of time. We can't solve it in polynomial time using linear approaches.Therefore, a method for solving the puzzle quickly will be derived that takes advantage of manycore architecture on modern and future computer systems which will bring down the time cost of solving an *NxN* Sudoku to a reasonable amount.

2. Sudoku and commonly used approaches for solving

Due to the large combinations, there are no serial algorithms that can solve sudoku in polynomial time. Therefore we have used heuristic approaches to fill up as many empty cells as possible.

We have used three types of heuristic strategies to try and fill in numbers on the board.

1. **Elimination** - This occurs when there is only one valid value for a cell.

2. **Lone Ranger** - This is a number that is valid for only one cell in a row, column, or box. There may be other numbers that are valid for that particular cell, but since that number cannot go anywhere else in the row, column, or box, it must be the value for that cell.

3. **Twins** - These are a pair of numbers that appear together twice in the same two cells and only in those two cells. When twins are found, all other numbers can be eliminated as possible values for that cell.

3. Proposed Algorithm

We have applied each strategy viz. Elimination, Lone-ranger & Twins , one at a time. If the given strategy makes a change to the valid values for any cell or sets the value of a cell, then we repeat the strategies starting from elimination.

Basically, after taking the input grid , the application of heuristics is done as follows till none of the heuristics make any change:

a. Sequence in which the heuristics are applied:
   Elimination -> Lone Ranger -> Twins

b. If the application of a heuristics causes some change, then the sequence is repeated from the beginning. This ensures that elimination and lone rangers being more useful are applied more frequently.

First, elimination is run in parallel across all boxes. Once it's done, then lone rangers will run in parallel across all boxes, and so on. If we find the value of a cell, then we remove that value from the list of possible values of the cells in the same row, column, and box.

We have assigned only some of the specific boxes to check the rows and the columns for the strategies so we don't do redundant checking.

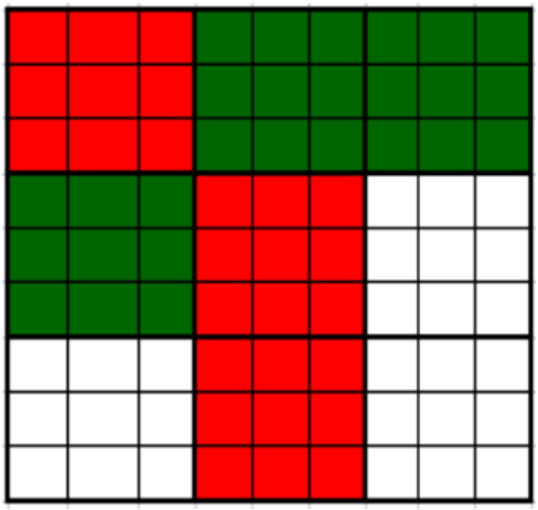


Fig 1. The red boxes check the rows and the green boxes check the columns.

If the heuristic approach returns a grid with unfilled cells left, then we pass it to the brute force; otherwise, we return the solution.

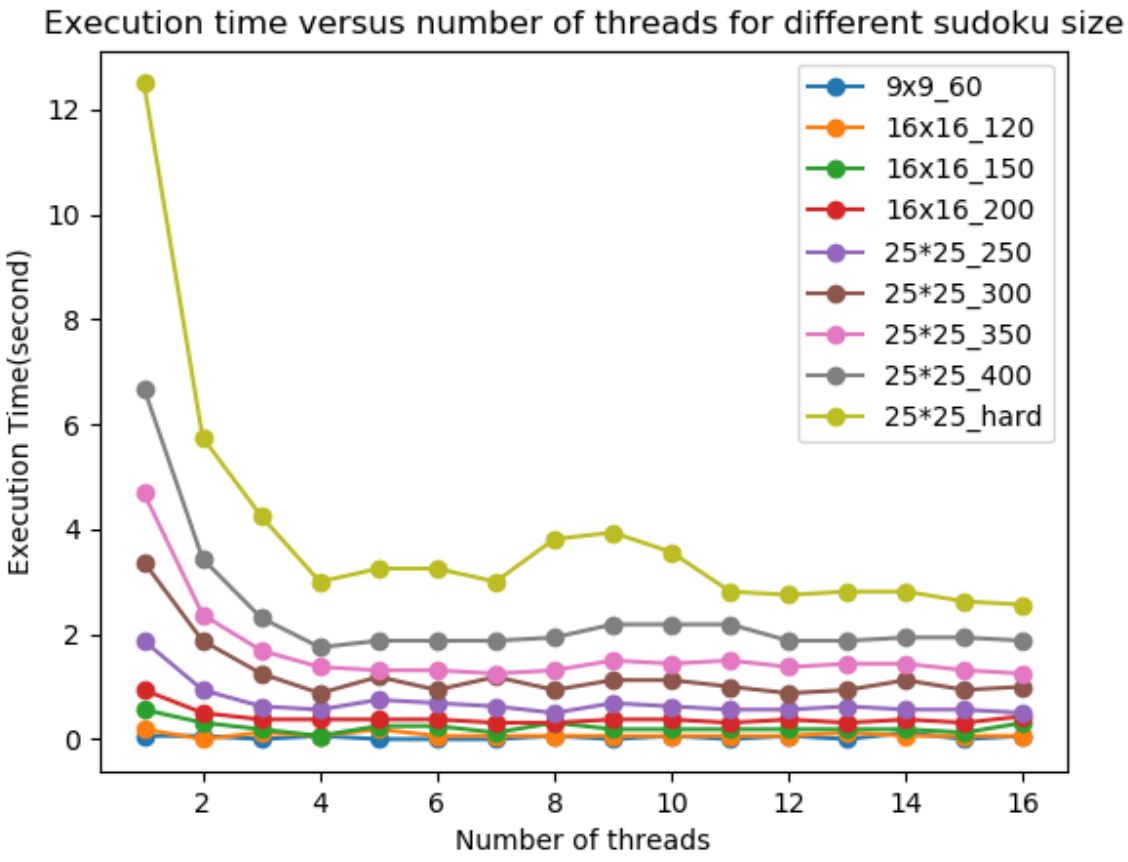
Brute Force Algorithm

Brute force algorithm uses depth first search approach. We have then created a list of grids for allotment among the available threads. Each thread gets a grid from the allotment list and uses a local stack to execute brute force DFS on it which is also parallelized. We have assigned the grid in round-robin fashion so that each thread gets equal share of processing to reduce time.
Brute force approach works in following way:

- Pop a grid from the search stack.
- Apply heuristics on it.
- Expand it by selecting the cell with the least number and pushing the new grids into the stack.
- Whenever the solution is found the thread sets the shared variable flag as true and all the threads exit the parallel section.

Finally, the main thread will return either the solution, or no solution if the stack is empty.

4. Timing Analysis



Note:

- 9x9_60 for instance means a sudoku of size 9x9, with 60 missing values to be filled with.
- Vertical axis represents time required to solve the sudoku & horizontal axis represent number of threads exploited to get the job done, i.e fill the missing entries such that the filled grid is a valid sudoku. nxn_y: y missing entries are filled in execution, t seconds using n threads for a sudoku of grid size nxn.
- Speed-up of ~4 is achieved on using 4 or greater threads and more or less converges for greater number of threads. For the application of solving the sudoku on the machine considered, 4 seems to be an ideal choice for running the algorithm considered. For threads less than 4, execution time suffers and for threads greater than 4, over utilisation of resources happen with marginal improvement in execution time.

5. Conclusion

- **Sudoku algorithm considered and implemented successfully.**
- **Concepts of "Elimination", "lone-ranger", "twins", etc were considered to solve the sudoku puzzle.**
- **OpenMP parallelisation framework implemented, and a speedup of ~4 is achieved on exploiting 4 or greater threads.**

6. Contribution Schema

Amit Kumar (16d070034)	Sudoku algorithm code, Presentation
Saurabh Kumar (16d100018)	OpenMP parallelization, Timing Analysis
Amit Kumar (16d100023)	Code-debugging, Report
Ali Imam (16d100017)	Sudoku algorithm code , Report

Fairly equal contributions in hindsight!!

7. System Details

Code has been run on the following machine for timing analysis purpose:

Hardware details:

- Local machine(Acer Nitro AN515-51)
- Processor: Intel(R) Core(TM) i5-7300HQ CPU @ 2.50GHz, 2496 Mhz, 4 Core(s), 4 Logical Processor(s)
- Installed RAM: 8.00 GB(7.89 GB usable)

OS details:

- Linux(WSL(Windows Subsystem for Linux) session on MobaXterm) installed in Windows 10 Home Single Language