Parallel Sudoku Solver

Made By:

Sameer Prajapati	Nirmal Rajput	Kiran Dapkar	Chaw Akameta Enling
170050061	170050046	170050020	170050054
Computer Science	Computer Science	Computer Science	Computer Science
IIT Bombay	IIT Bombay	IIT Bombay	IIT Bombay

Guided By:

Prof. Shivasubramanian Gopalakrishnan

IIT Bombay

Course: High Performance Scientific Computing (ME 766)

Abstract:

Sudoku is a logic-based, combinatorial number-placement puzzle having n^2xn^2 grids of nxn blocks that becomes increasingly difficult due to combinatorial explosion. As a result, a sequential implementation of a Sudoku puzzle solver can become both data- and compute-intensive. In this project, we proposed a new parallel algorithm solver for the generalized NxN Sudoku grid that utilizes the manycore architecture. Our test results show that the algorithm scales well as the number of cores grows.

About sudoku:

Standard Sudoku is a logic-based number puzzle featuring a 9x9 board divided into rows, columns, and 3x3 square blocks. The goal of the game is to fill each row, column, and block with unique numbers in the range 1 through 9. Variations of sudoku have used larger boards, such as hexadoku: sudoku with a 16x16 board. Our code for solving problems is also versatile; that is, it can take input to a 16x16 board as well as a 9x9 board.

Here is an example of a sudoku puzzle (left) along with its solution (right):

	1			8			2	
5								1
	2	4	1		3	5	7	
1								5
		5	8		9	4		
9								7
	5	9	4		2	1	8	
4								2
	6			1			4	

3	1	7	5	8	6	9	2	4
5	9	8	7	2	4	3	6	1
6	2	4	1	9	3	5	7	8
1	8	3	6	4	7	2	9	5
2	7	5	8	3	9	4	1	6
9	4	6	2	5	1	8	3	7
7	5	9	4	6	2	1	8	3
4	3	1	9	7	8	6	5	2
8	6	2	3	1	5	7	4	9

All Sudoku puzzles will have some given numbers on them, and the difficulty of the puzzle varies depending on how many numbers are given, as well as the location of the numbers that are given.

Computational Complexity:

From a mathematical perspective, it has been proven that the total number of valid Sudoku grids is **6,670,903,752,021,072,936,960** or approximately **6.671×10^{21}**. Even if the player is given around 20 numbers, that still leaves a lot of solutions that need to be permuted in order to find the one correct solution. In particular, the basic way of solving Sudoku puzzles is to look for specific patterns, and having parallelism can allow the program to search for these patterns much more quickly, as they can be searched in parallel due to the fact that they're not dependent on each other.

Existing Approaches:

Total number of valid Sudoku puzzles is approximately 6.671×1021(9*9 matrix). So, some of the techniques currently in use are:

- Brute force technique - long time
- Simulated Annealing - only easy problems
- Linear system approach - hard to solve with fewer clues
- Artificial bee colony algorithm: division into 2 sets
- SAC Algorithm: Assigning values to each cell and backtracking

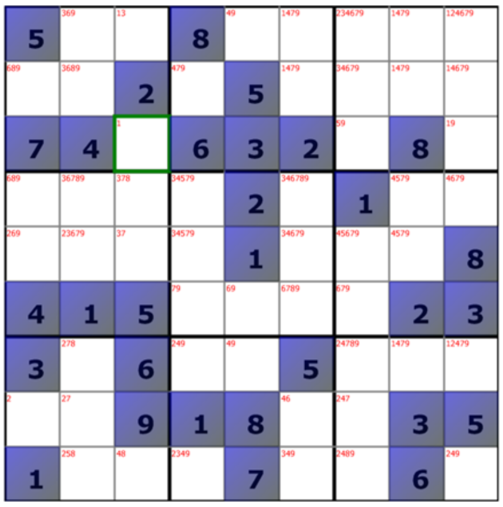
Proposed Approach:

Solving sudoku is proven to be an NP-complete problem. No serial algorithms that can solve sudoku in polynomial time. We will use a humanistic algorithm to fill up as many empty cells as possible. This algorithm doesn't guarantee a solution. If necessary, the brute force algorithm solves the rest of the puzzle. Four types of strategies to try and fill in numbers on the board are explained below.

Sudoku Solving Techniques:

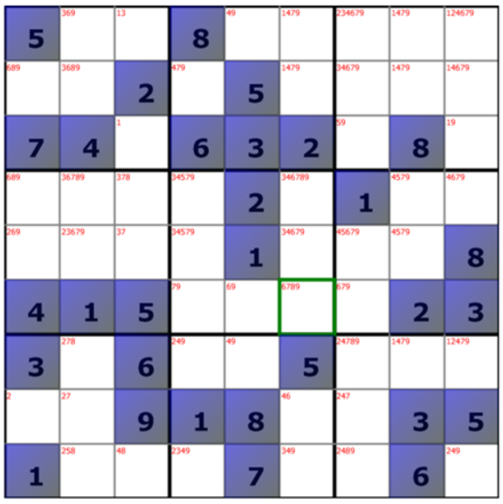
1. Elimination:

   Elimination technique is probably one of the simplest ways to start solving a Sudoku grid. Basically this method scans the grid from the top left cell to the bottom right cell going one row at a time and looks for each cell, it checks whether the number of possible values is equal to 1 then it assigns the single possible number to the cell. This method is straightforward and is



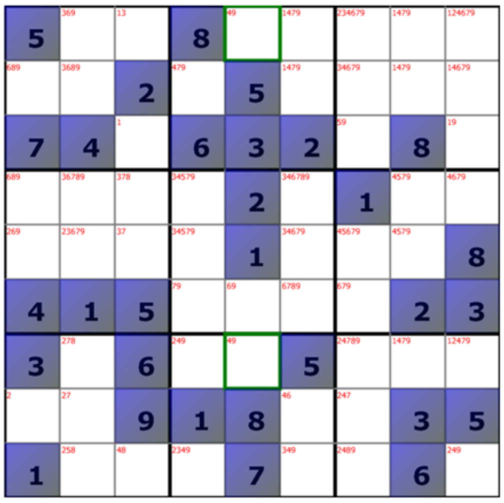
a basic step when deciding on which value to fill in a cell. The figure below shows the highlighted cell in green which could be assigned one possible value that happens to be 1 too since values 2,4,5,7 are in the same minigrid. 3, 6, 8 are on the same row and 9 is on the same row.

2. Finding Lone Rangers:



A Lone Ranger is a number that is one of multiple values for a cell but it appears only once in a row, column or minigrid. For example, the green cell shown in the figure below has 4 possible values. Intuitively, a human solving this grid will most likely skip this cell because it seems "too early to predict its value" because of the several possible values. But if we take a closer look at the cell's row, note that the number 8 appears only once in the row and it appears as a possible value for that cell. Hence, 8 is indeed the correct value for that cell.

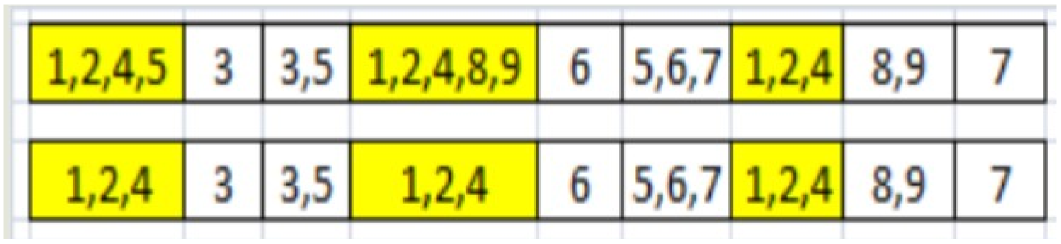
3. Finding Twins:



Twins are two cells that contain two numbers that appear nowhere else but in these cells on the same row, column or minigrid. For example, take a look at the two green cells below, both have the same two possible values 4 and 9 on the same column. This means that if one of the cells were to be assigned the value 4, the other cell would be assigned the value 9 and vice versa. Therefore we can be confident that 4 and 9 will not be assigned to any other cell on the same column. Therefore we remove the value 9 from the remaining cell in the same column that had two possible values 6 and 9 so that cell has only value 6 left. By process of elimination 6 will be assigned to that cell.

4. Finding Triplets:

Triplets follow the same rules as twins, except with three values over three cells.

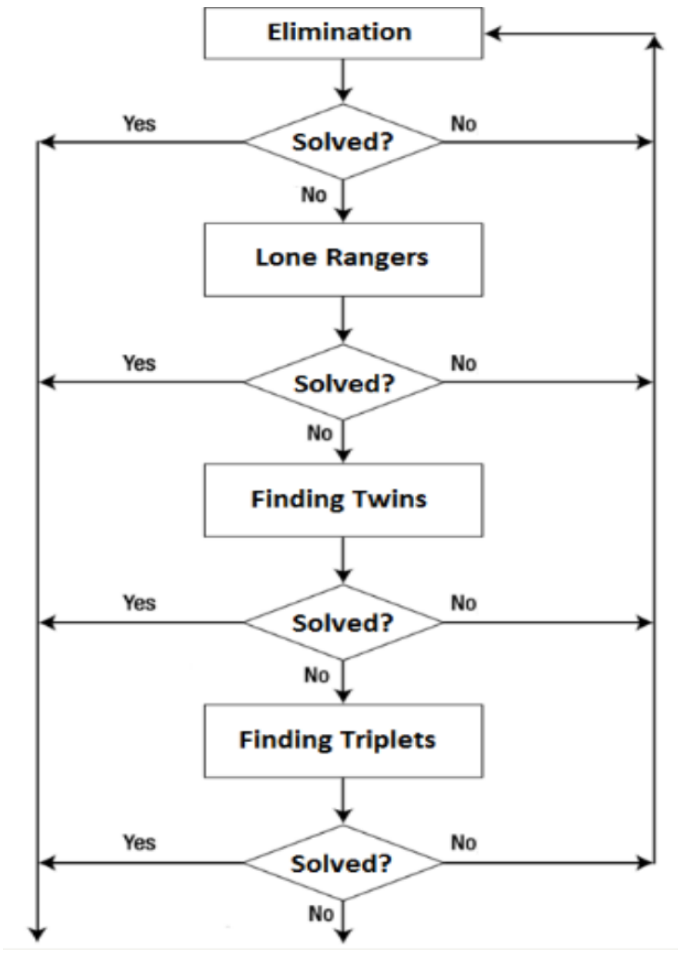


1,2,4,5	3	3,5	1,2,4,8,9	6	5,6,7	1,2,4	8,9	7
1,2,4	3	3,5	1,2,4	6	5,6,7	1,2,4	8,9	7

The top row depicts a row of a sudoku board before triplets are applied. The bottom row is the resulting valid values after triplets have been applied. In the top row, 1, 2, and 4 only appear three times in the row and they all appear in the exact same three cells, so they are triplets.

Brute-Force Elimination

When all logical means have been used to solve a Sudoku puzzle and the puzzle remains unsolved, we have to perform some guesswork and choose a value for a cell and see if that helps to solve a puzzle but which cell? As a heuristic I have decided to choose the cell with the least number of possible values. This will decrease the probability of guessing it wrong.I should also mention the possibility of a wrong guessing, so what happens if the algorithm guessed a value that turned out to yield an unsolvable puzzle within a few steps. To work around the problem, we introduce the notion of "backtracking" or simply, undoing the puzzle to a previous state just before we took the wrong guess and continue from there by taking a different guess. To get the undo strategy working, we will utilize the classic stack data structure. When a guess is made, a copy of the puzzle is pushed onto the stack before the guess is taken. When a dead end is reached, the puzzle is popped from the stack and replaces the current unsolvable one.



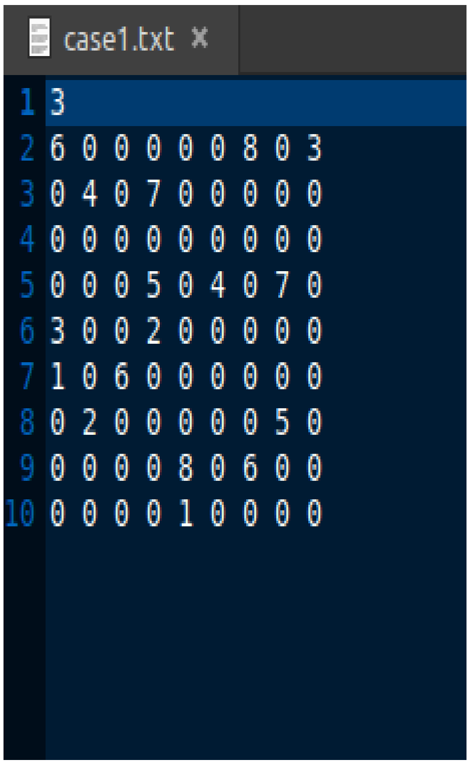
We can depict from the flowchart that each strategy is applied one at a time. If a strategy makes a change to the valid values for any cell or sets the value of a cell, then we repeat the strategies starting from elimination. If the strategy makes no change, then we move on to the next strategy. Elimination and lone ranger can be applied more often since those are the strategies most likely to make changes to the sudoku puzzle. 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.

Parallelization :

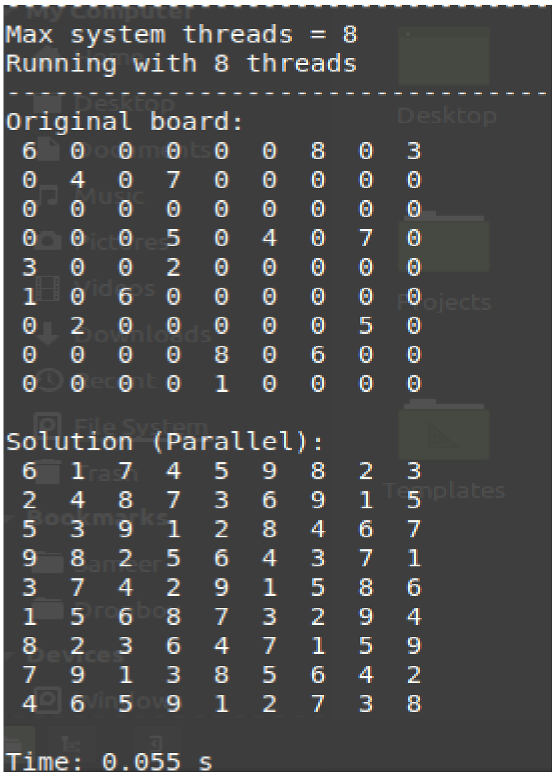
- ➡ Parallelizing by box for each strategy.
  - First, elimination is run in parallel across all boxes.
  - Once it's done, the lone rangers will run in parallel across all boxes, and so on.
  - We assign only specific boxes to check the rows and the columns for the strategies so we don't do duplicate work.
  - For example, we don't want two boxes in the same rows to check for elimination in the same three rows.
  - We also choose the boxes in such a way that we don't have one box doing extra work by checking both columns and rows.
- ➡ Brute force algorithm uses depth first search approach.
  - First, we fill in the first empty cell with the smallest valid number.
  - Then, we try to fill in the next empty cell in the same way.
  - If we reach an empty cell that does not have any valid numbers, we backtrack to the most recently filled cell and increment its number by 1.
  - If the number cannot be incremented, then we backtrack again.
  - We continue doing this until there are no more empty cells on the board.
  - This means we found the solution and we return it. Or we backtracked to the first unfilled cell.
  - In this case, no solution exists for the board. We then combine our serial brute force algorithm with a shared stack.
  - We create a board for all permutations of valid numbers of the first 7 cells and fill in those 7 cells with the permutations. We then push all those boards onto the stack.
  - Next, multiple threads pop from the stack in parallel and try to find a solution using brute force. If a partial board popped from the stack isn't the solution, the board is discarded and the thread pops another one off the stack.
  - The first thread to find a solution will abort all other threads.
  - Finally, the main thread will return either the solution, or no solution if the stack is empty.

Input/Output:

Input is N, the dimension of the grid (minigirdsize) of the board. Then the board itself is passed as input. The boards are represented as a 2-D N^2xN^2 array of integers where the 0's represent the empty squares.
Example can be seen in figure.



```
3
6 0 0 0 0 0 8 0 3
0 4 0 7 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 5 0 4 0 7 0
3 0 0 2 0 0 0 0 0
1 0 6 0 0 0 0 0 0
0 2 0 0 0 0 0 5 0
0 0 0 0 8 0 6 0 0
0 0 0 0 1 0 0 0 0
```



```
Max system threads = 8
Running with 8 threads
-----------------------------------
Original board:
 6   0   0   0   0   0   8   0   3
 0   4   0   7   0   0   0   0   0
 0   0   0   0   0   0   0   0   0
 0   0   0   5   0   4   0   7   0
 3   0   0   2   0   0   0   0   0
 1   0   6   0   0   0   0   0   0
 0   2   0   0   0   0   0   5   0
 0   0   0   0   8   0   6   0   0
 0   0   0   0   1   0   0   0   0

Solution (Parallel):
 6   1   7   4   5   9   8   2   3
 2   4   8   7   3   6   9   1   5
 5   3   9   1   2   8   4   6   7
 9   8   2   5   6   4   3   7   1
 3   7   4   2   9   1   5   8   6
 1   5   6   8   7   3   2   9   4
 8   2   3   6   4   7   1   5   9
 7   9   1   3   8   5   6   4   2
 4   6   5   9   1   2   7   3   8

Time: 0.055 s
```

Evaluation Methodology:

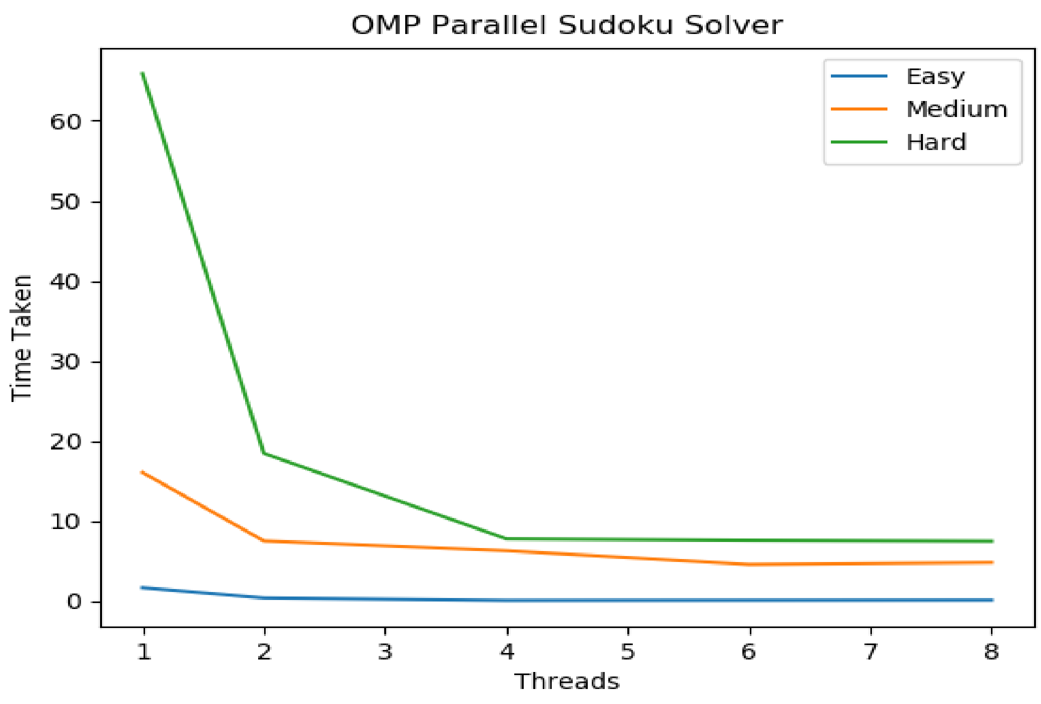
We performed our evaluation on the Intel Octa Core 2.66 GHz machine with 8 GB of RAM running the Debian Linux kernel 2.6.32. Due to the non-deterministic nature of multithreaded executions, run-times can vary even when solving the same puzzle. We therefore averaged the results of 10 9x9 puzzles, running them 5 times for each variant of our algorithms. We measure elapsed time using the CycleTimer method which gives us accuracy to the microseconds.

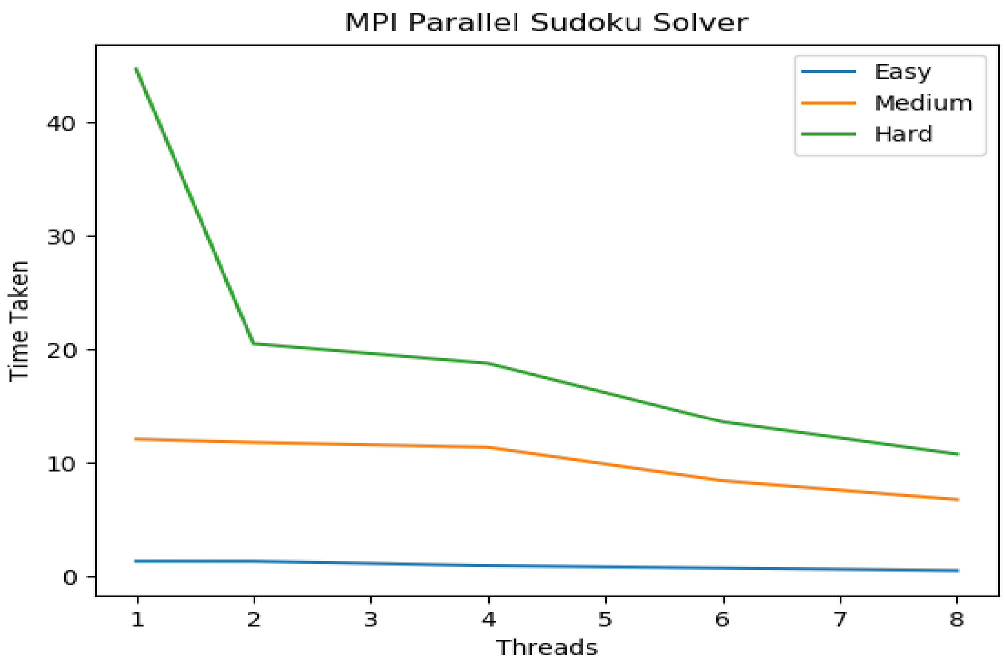
Machine Description :
RAM 8GB
Intel i5 8th Gen, 8 Core
Graphic Card - Nvidia Geforce MX150, 2GB
Linux Mint 20.1 (base: Ubuntu 20.04)

Evaluation Results:

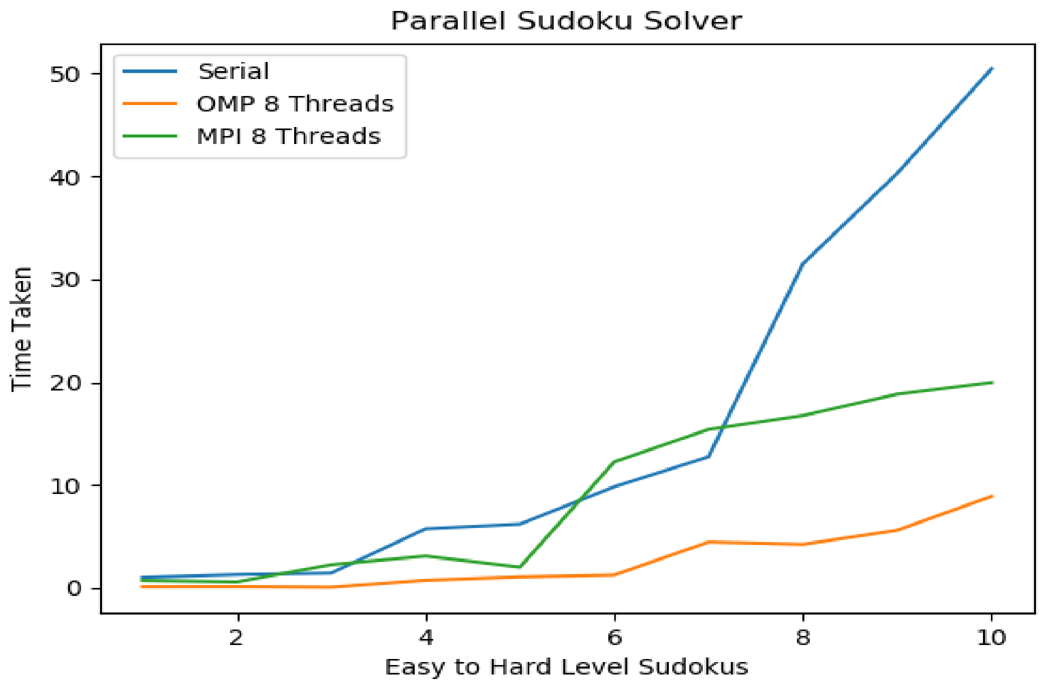
We tested out results on 9x9 and 16x16 puzzles with different levels of hardness. We included three levels- Easy, Medium and Hard which gave us significantly clearer results. The fewer entries the more difficult the sudoku becomes.
The graph for the same can be found below: (Time Taken is in ms).

Thread 4 was giving more consistent performance. For more threads overall speedup was not much significant since they also required a fair amount of data share.





Running 10 sudokus from easy to hard level with all the implemented solver.



Conclusion:

We proposed an implementation of a Sudoku puzzle solver and a reduction in execution time through parallelization. We created the parallel version, which allowed threads to work together on the same branch in graph to quickly explore its depth. As a result, there was a significant speedup in the algorithm, giving us reduced time as the number of cores grows.

References:

- Parallel Depth-First Sudoku Solver Algorithm https://alitarhini.wordpress.com/2011/04/06/parallel-depth-first-sudoku-solver-algorithm/
- Brute-force of Sudoku solving algorithms http://en.wikipedia.org/wiki/Sudoku_solving_algorithms#Brute-force_algorithm
- https://alitarhini.wordpress.com/2012/02/27/parallel-sudoku-solver-algorithm/