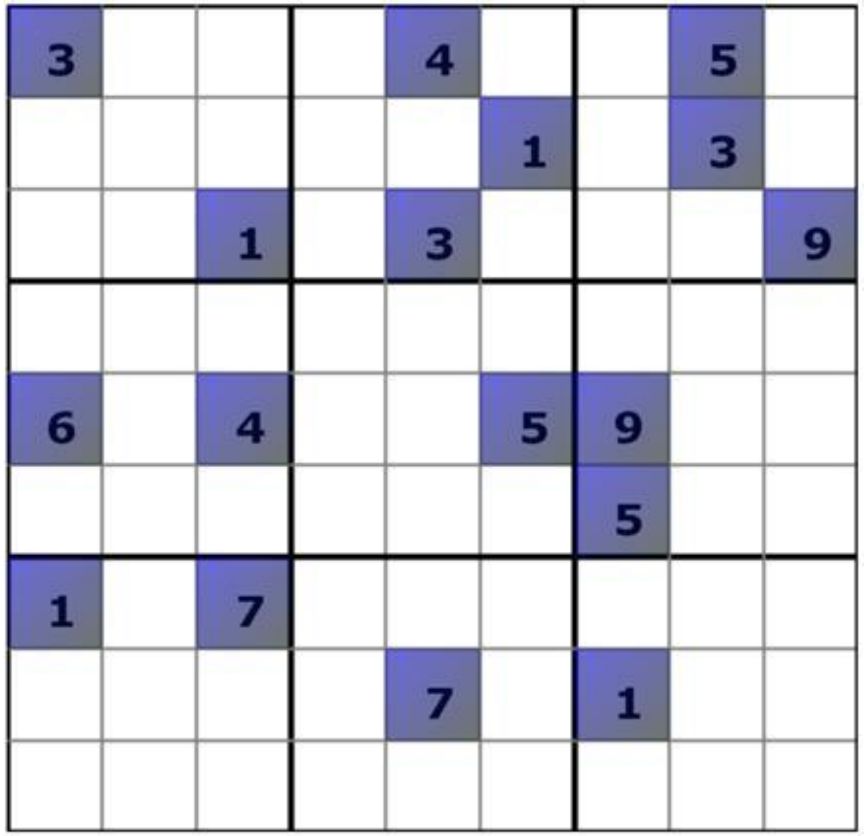
PARALLELIZED SUDOKU SOLVING ALGORITHM USING OpenMP

Sudoku: the puzzle



3				4			5	
					1		3	
		1		3				9
6		4			5	9		
						5		
1		7						
				7		1		

- A standard Sudoku puzzles contains 81 grids :9 rows and 9 columns.
- 9 non- overlapping blocks, each block consists of 3 rows and 3 columns.
- Many strategies to solve Sudoku puzzles; solutions might not be unique, or might not even exist!
- The difficulty in solving is relative to the number of indices provided

Commonly used approaches

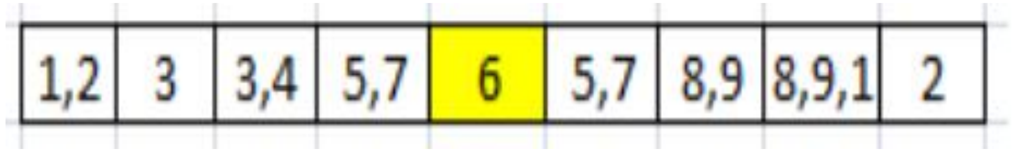
- Total number of valid Sudoku puzzles is approximately **6.671×10^{21} (9*9 matrix)**
- Brute force technique- long time
- Simulated Annealing- only easy problems
- Linear system approach- hard to solve with fewer clues
- Artificial bee colony algorithm: division into 2 sets
- SAC Algorithm: Assigning values to each cell and backtracking

PROPOSED ALGORITHM

- Solving sudoku is proven to be an NP-complete problem(**NP-complete problem**, any of a class of computational problems for which no efficient solution algorithm has been found.)
- No serial algorithms that can solve sudoku in polynomial time.
- Use of 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.

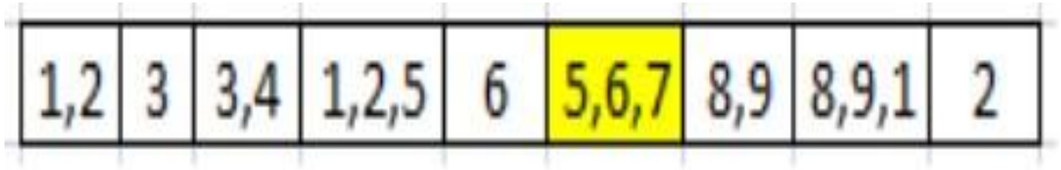
PROPOSED ALGORITHM- 4 STRATEGIES

- **ELIMINATION:** This occurs when there is only one valid value for a cell. Therefore, that value must belong in the cell.



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

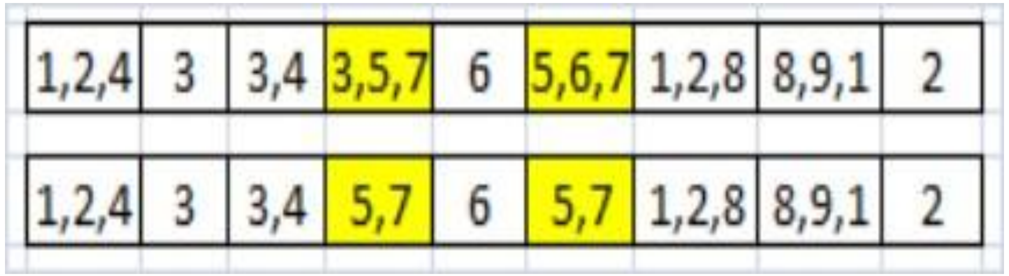
- **LONE RANGER:** This is a number that is valid for only one cell in a row, column, or box. There may be other numbers that are valid for that particular cell, but since that number cannot go anywhere else in the row, column, or box, it must be the value for that cell.



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

PROPOSED ALGORITHM- 4 STRATEGIES

- **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.

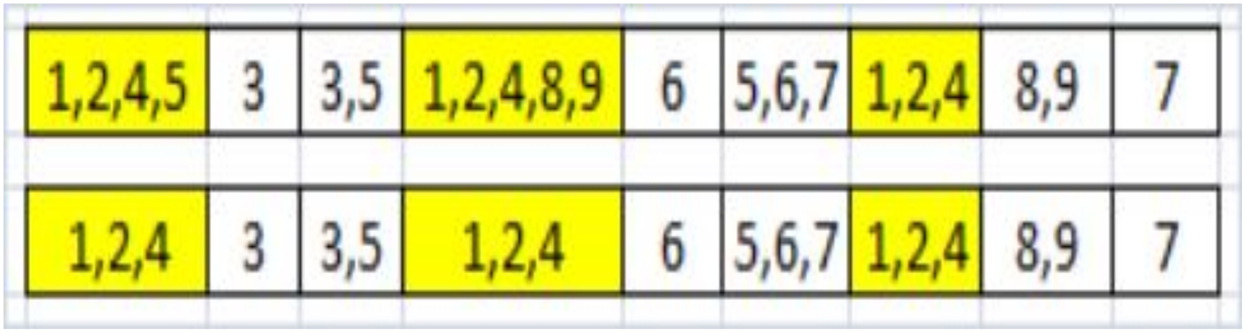


1,2,4	3	3,4	3,5,7	6	5,6,7	1,2,8	8,9,1	2
1,2,4	3	3,4	5,7	6	5,7	1,2,8	8,9,1	2

The top row depicts a row where the highlighted cells contain twins. The bottom row shows the possible values left for those cells after the twins rule has been applied. In the top row, 5 and 7 show up together twice and only twice in the row. Because they are twins, the 3 and 6 that were valid values for those highlighted cells can now be eliminated as possible values.

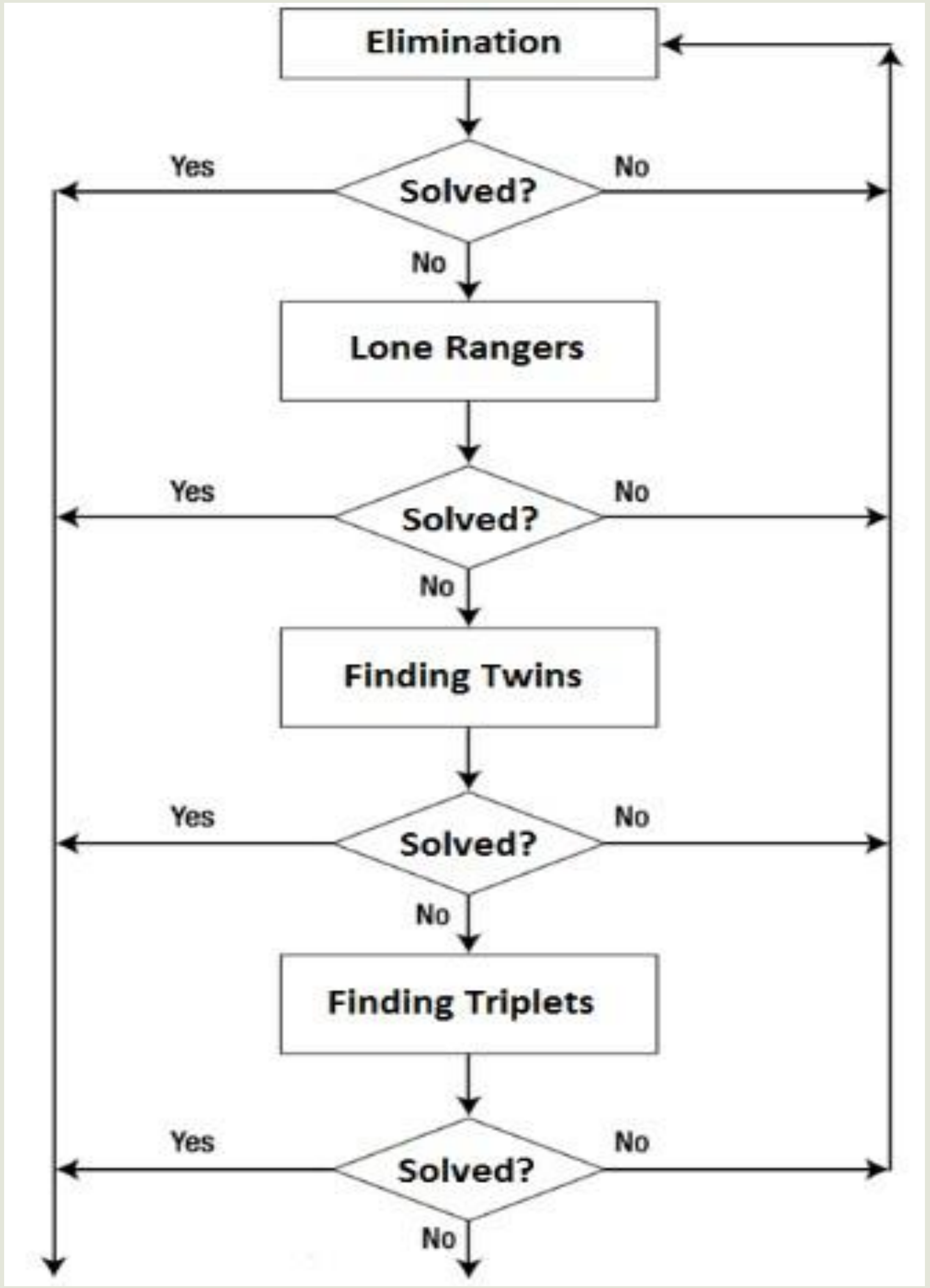
PROPOSED ALGORITHM- 4 STRATEGIES

- **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 is applied. The bottom row is the resulting valid values after triplets has 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.


Elimination
Solved?
Yes
No
No
Lone Rangers
Solved?
Yes
No
No
Finding Twins
Solved?
Yes
No
No
Finding Triplets
Solved?
Yes
No
No

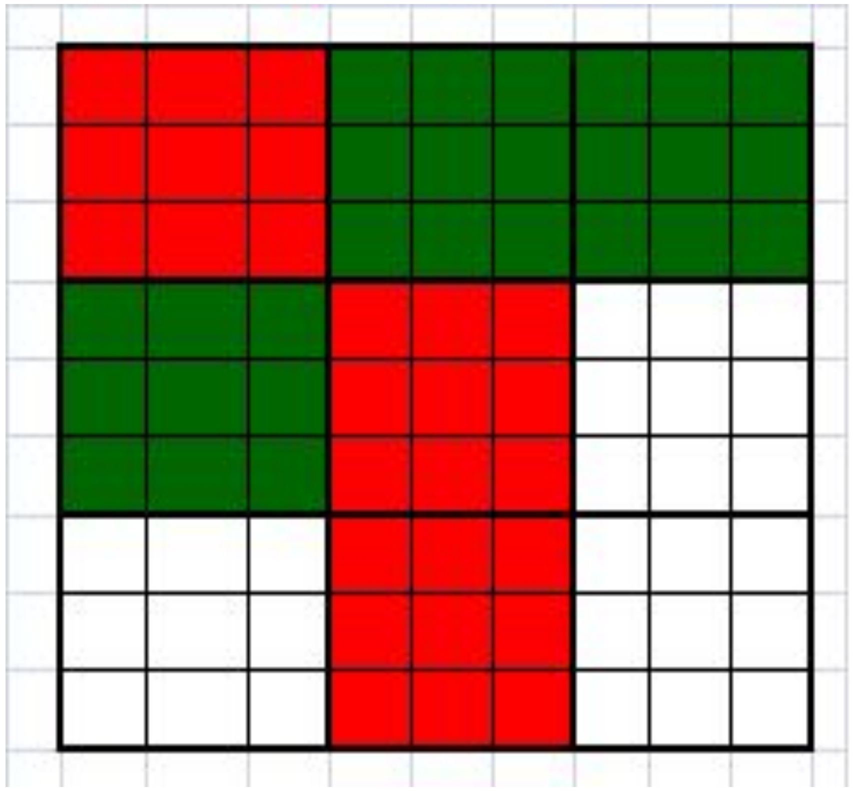
PROPOSED ALGORITHM

- 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.

PROPOSED ALGORITHM

- Parallelizing by box for each strategy.
- First, elimination is run in parallel across all boxes. Once its done, then 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

PROPOSED ALGORITHM



- The red boxes check the rows and the green boxes check the columns.
- If the humanistic algorithm returns a board with unfilled cells left, then we pass it to the brute force.
- Otherwise, we return the solution.

PROPOSED ALGORITHM

- **B**rute 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 either 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.

PROPOSED ALGORITHM

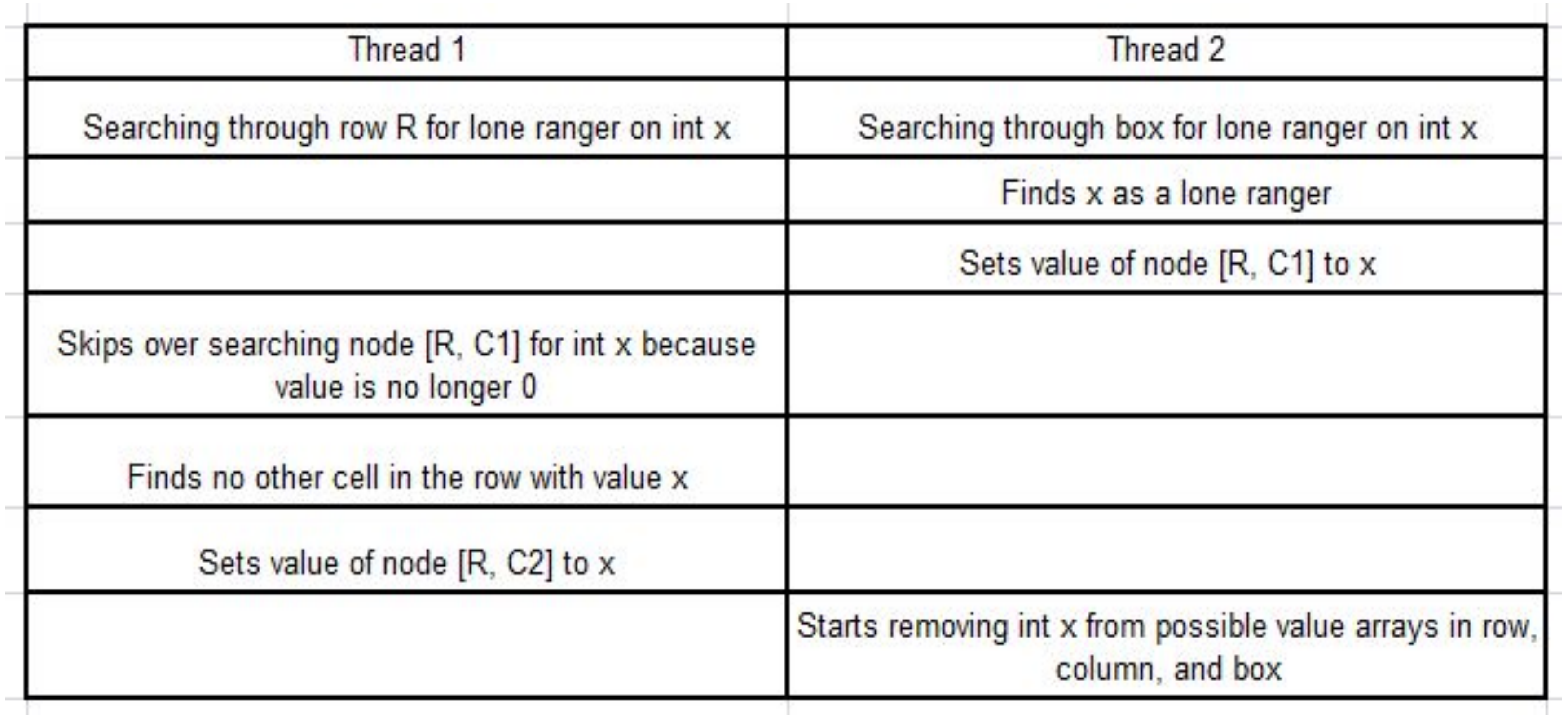
- We then combine our serial brute force algorithm with shared stack.
- We create a board for all permutations of valid numbers of the first 7 cells and fill in those 7 cells with the permutations. We then push all those boards onto the stack.
- Next, multiple threads pop from the stack in parallel and 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.

IMPROVIZATIONS

- Workload sizes for each thread were too small since we were finding valid numbers for only one cell, which is not an intensive nor time-consuming task. Each thread ended up finishing quickly and was waiting to gain access to the stack.
- Instead of only filling in the first empty cell, we fill in the first 7 empty cells. We push all possibilities of the first 7 being filled to the stack. Then threads pop the copies from the stack in parallel.
- Chose to fill in 7 cells because, it was found to give a good execution time in comparison to other numbers of cells.
- Increased the workload size for each thread, which helped in lowering the contention on the stack.

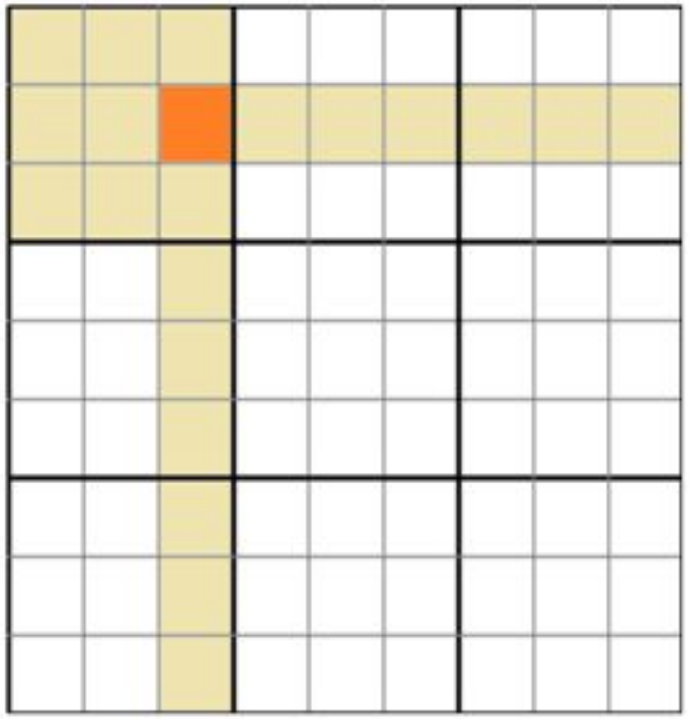
RACE CONDITION

- Added a lock because the algorithm ran into the following race condition:



Thread 1	Thread 2
Searching through row R for lone ranger on int x	Searching through box for lone ranger on int x
	Finds x as a lone ranger
	Sets value of node [R, C1] to x
Skips over searching node [R, C1] for int x because value is no longer 0	
Finds no other cell in the row with value x	
Sets value of node [R, C2] to x	
	Starts removing int x from possible value arrays in row, column, and box

LOCK OPERATION

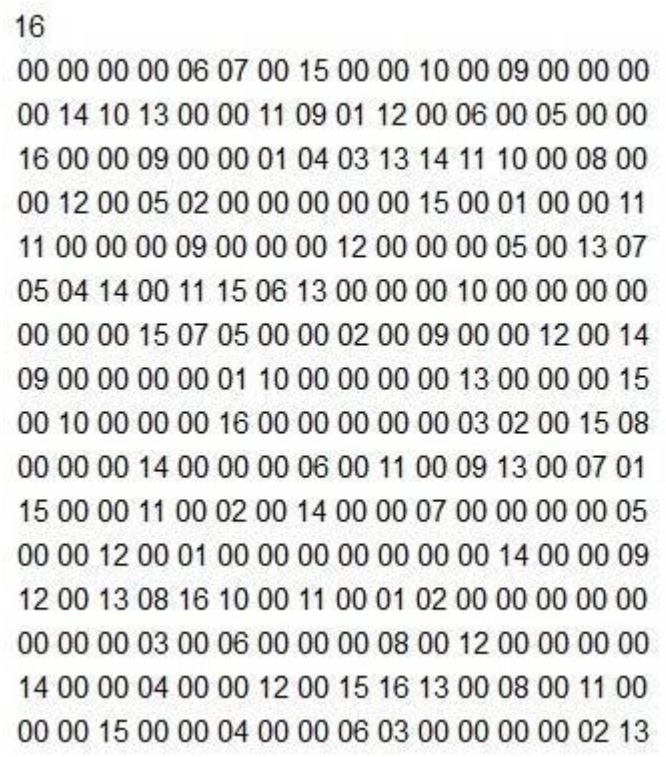


- LOCK is an operation that freezes the entire row, column and mini-grid, so that no other core could lock any cell within the same area. This area is the conflict *boundary* of a cell.
- The shaded area represents the locked region which is the conflict boundary required to be locked in order to update cell C(2,3).

LOCK OPERATION

- If a core is unable to place a lock, due to a lock that had already been placed by a different core on another cell on the same conflict boundary, then the core chooses a different work item to process.
- This process reduces the idle time for a core, since it eliminates the need for waiting until the other core releases its lock on the conflict boundary, hence maximizing the overall efficiency of the algorithm.
- In this way, all the cells are filled with the values without running into race conditions.

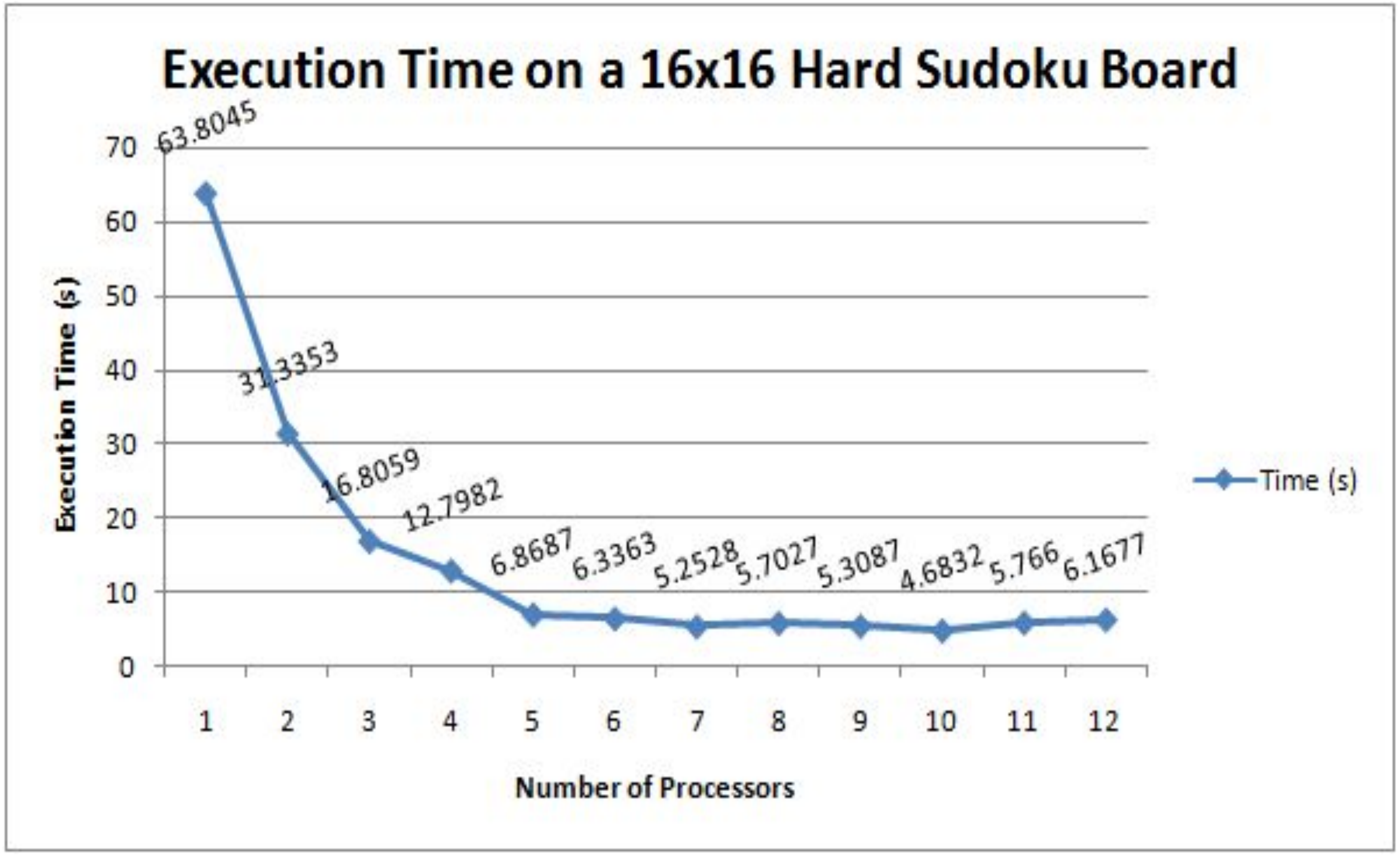
INPUT



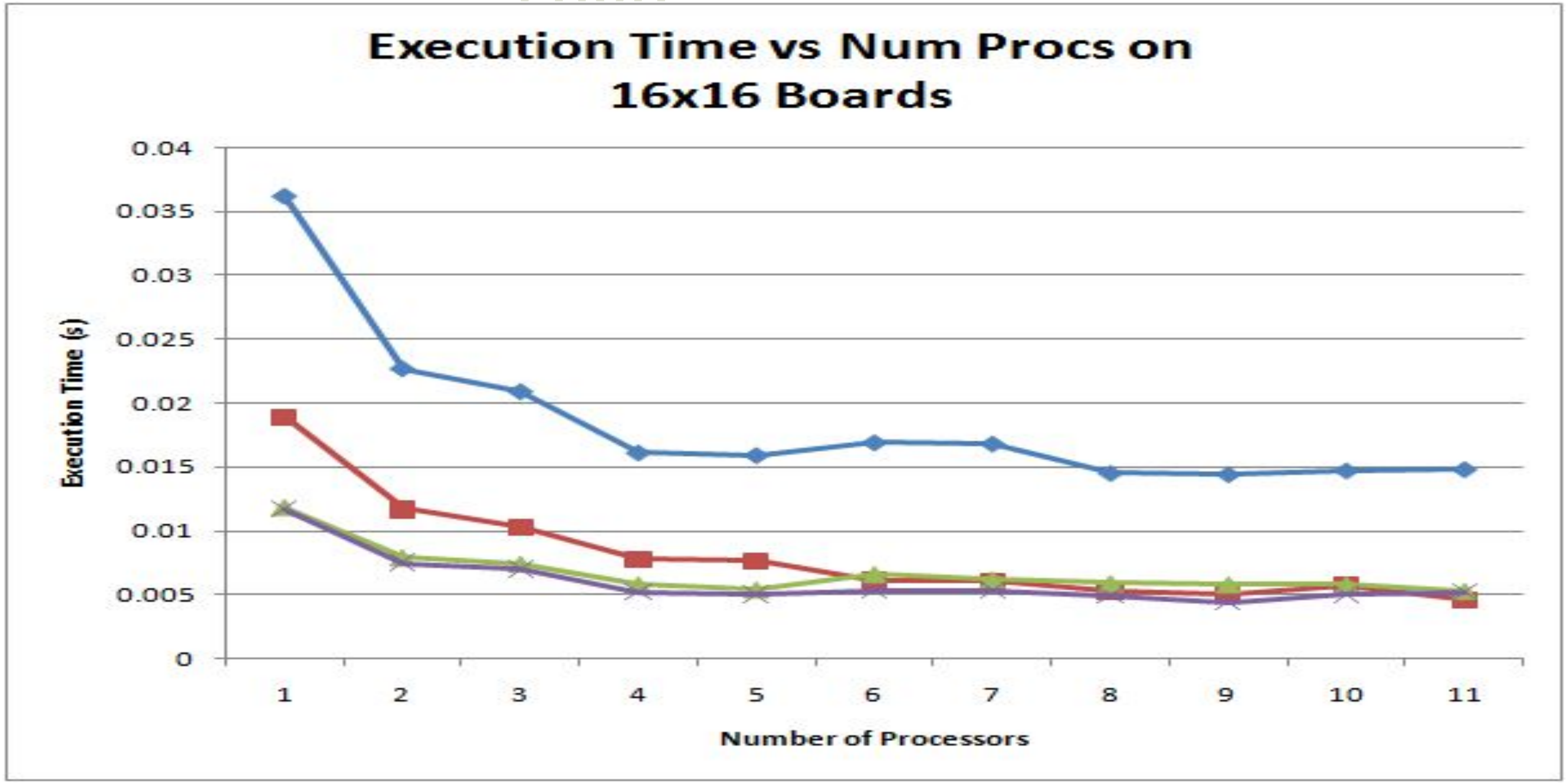
```
16
00 00 00 00 06 07 00 15 00 00 10 00 09 00 00 00
00 14 10 13 00 00 11 09 01 12 00 06 00 05 00 00
16 00 00 09 00 00 01 04 03 13 14 11 10 00 08 00
00 12 00 05 02 00 00 00 00 00 15 00 01 00 00 11
11 00 00 00 09 00 00 00 12 00 00 00 05 00 13 07
05 04 14 00 11 15 06 13 00 00 00 10 00 00 00 00
00 00 00 15 07 05 00 00 02 00 09 00 00 12 00 14
09 00 00 00 00 01 10 00 00 00 00 13 00 00 00 15
00 10 00 00 00 16 00 00 00 00 00 03 02 00 15 08
00 00 00 14 00 00 00 06 00 11 00 09 13 00 07 01
15 00 00 11 00 02 00 14 00 00 07 00 00 00 00 05
00 00 12 00 01 00 00 00 00 00 00 00 14 00 00 09
12 00 13 08 16 10 00 11 00 01 02 00 00 00 00 00
00 00 00 03 00 06 00 00 00 08 00 12 00 00 00 00
14 00 00 04 00 00 12 00 15 16 13 00 08 00 11 00
00 00 15 00 00 04 00 00 06 03 00 00 00 00 02 13
```

- Input: N, the dimension of the board. Then board itself is passed as input.
- The boards are represented as a 2-D NxN array of integers where the 0’s represent the empty squares.

BRUTE FORCE ALGORITHM-EXECUTION TIME



HUMANISTIC and BRUTE FORCE ALGORITHM- EXECUTION TIME



CONCLUSION

- The humanistic approach doesn’t guarantee a solution but it achieves a much better overall time.
- The brute force approach parallelizes well and guarantees to find a solution if one exists but it has a slow overall time.
- By combining both, we have a solver that is guaranteed to find a solution in a very short amount of time.

REFERENCES

- http://alitarhini.wordpress.com/2011/04/06/parallel-depth-first-sudoku-solver-algorithm/?relatedposts_exclude=372
- http://alitarhini.wordpress.com/2012/02/27/parallel-sudoku-solver-algorithm/
- http://en.wikipedia.org/wiki/Sudoku_solving_algorithms#Brute-force_algorithm
- http://www.andrew.cmu.edu/user/hmhuang/project_template/finalreport.html
- Miller, Russ and Laurence Boxer. Algorithms Sequential and Parallel: A Unified Approach. Hingham, MA: Charles River Media, 2005. Print.