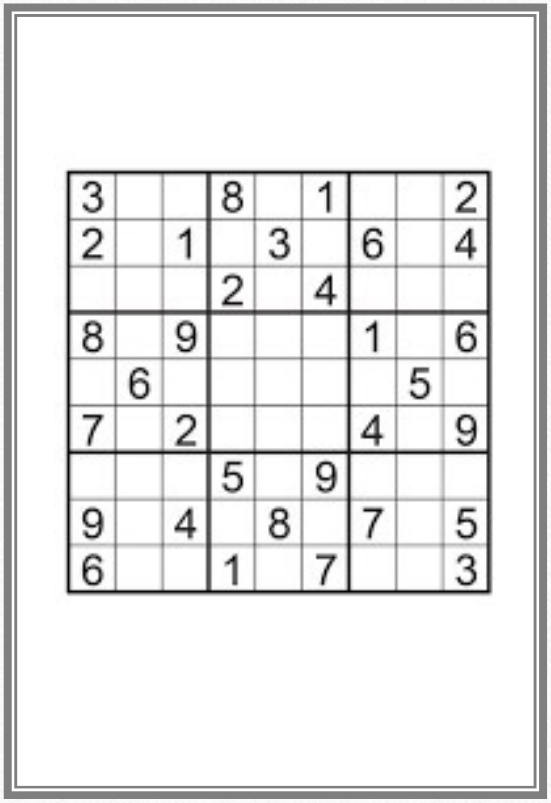


3			8		1			2
2		1		3		6		4
			2		4			
8		9				1		6
	6						5	
7		2				4		9
			5		9			
9		4		8		7		5
6			1		7			3

Sudoku Algorithm using Parallel Processing

Hemchand Kalugotla

Agenda

What is Sudoku

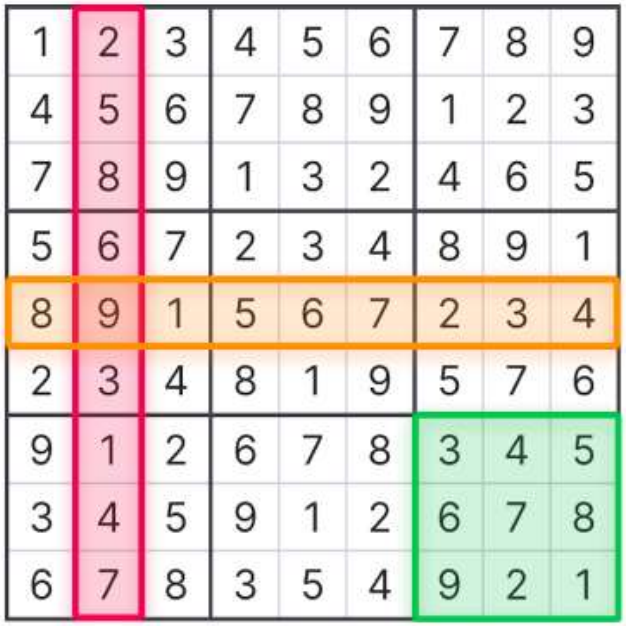
Ways to Solve and Parallel approaches

Challenges

Approaches Used

Results

What is Sudoku?



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

- Sudoku is a logic-based number placement puzzle.
- Sudoku requires no calculation or arithmetic skills. It is essentially a game of placing number in squares.
- The classic game is played on a 9×9 grid which contains:
  - 9 non-overlapping blocks.
  - Each block consists of 3 rows and 3 columns.
- The goal of the game is to fill each row, column and subsection with numbers from 1 to 9.
  - Each sector of 3×3 cells must contain numbers from 1 to 9 without repeats.
  - Each vertical column must contain numbers from 1 to 9 without repeats.
  - Each horizontal row must contain numbers from 1 to 9 without repeats.

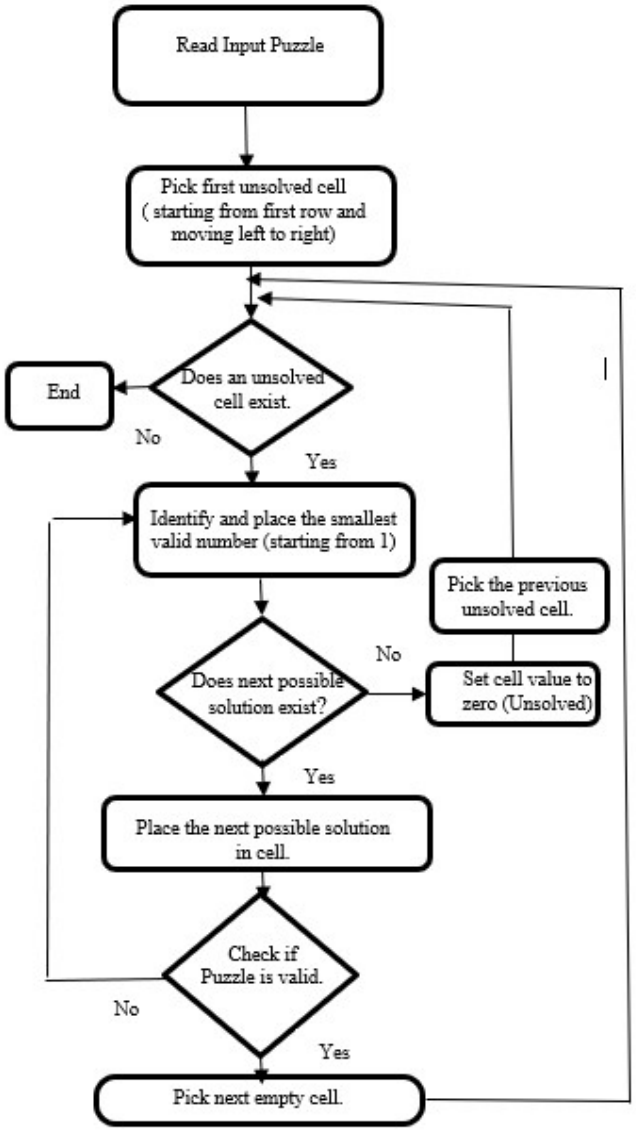
Ways to Solve

- Total number of valid Sudoku puzzles is approximately 6.671×1021 (9*9 matrix)
- The difficulty in solving is relative to the number of indices provided
- Many strategies to solve Sudoku puzzles
  - Boltzmann Machine
  - Rule-based
    - Similar to how humans solve
  - Backtracking
    - Guessing number in each cell.
    - Backtrack and try a different number if dead end is reached.
- Algorithm selected for project (Backtracking)
  - Guaranteed solution if input puzzle is valid
  - Exhaustive search involving lot of iterations
    - Reduce number of iterations/time to solve with parallel approach

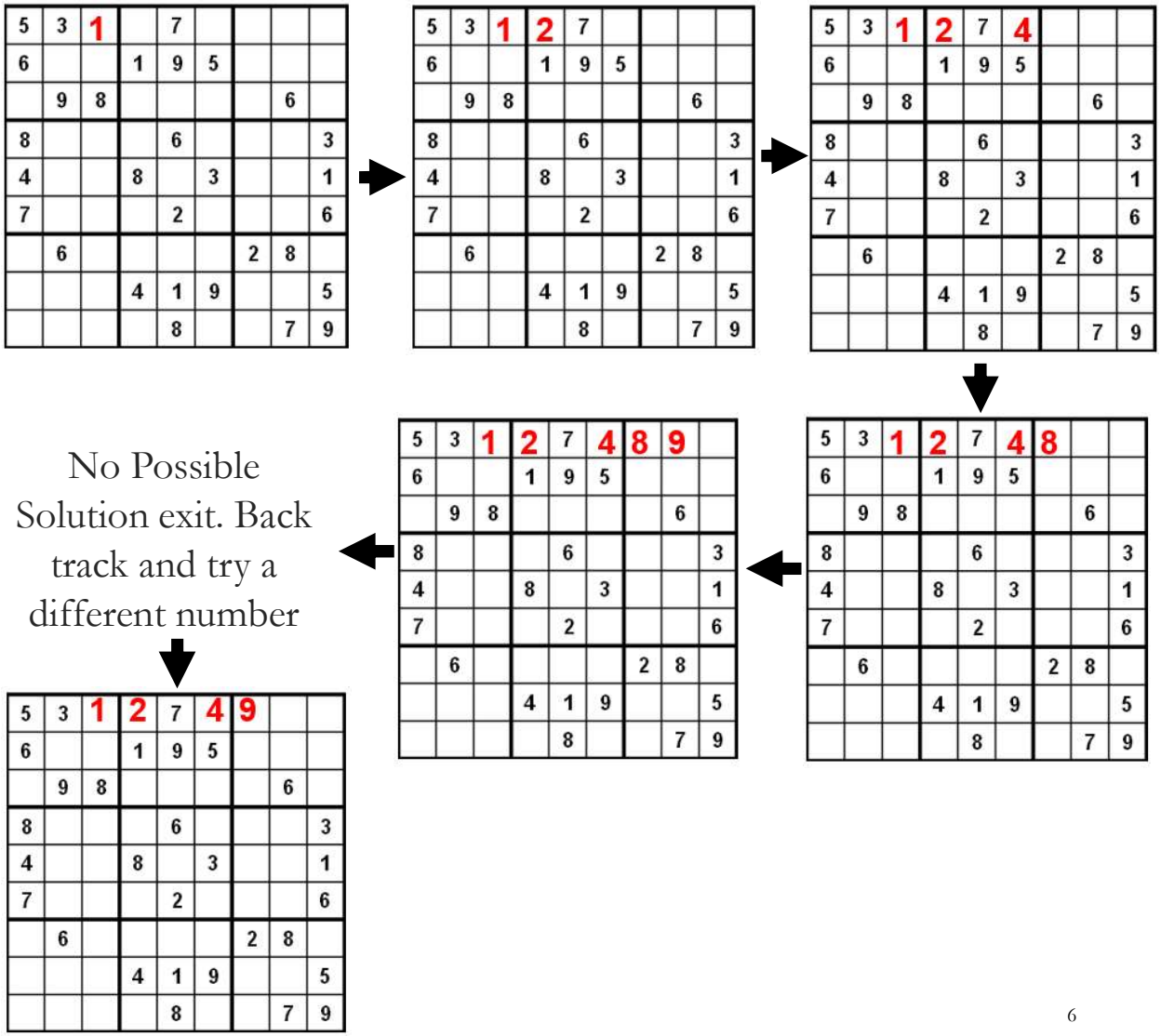
Back Tracking Algorithm

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

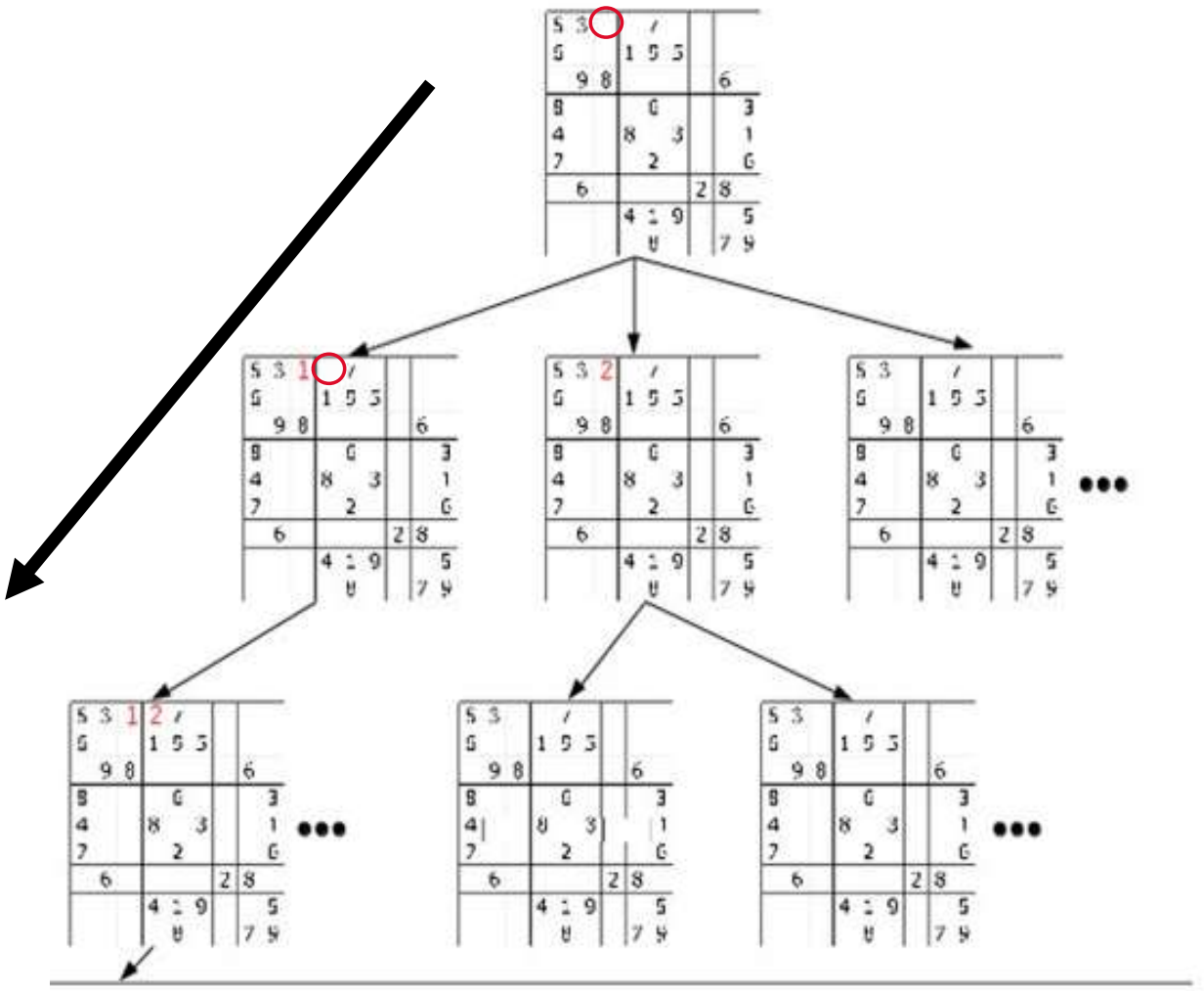
- First, fill in the first empty cell with the smallest valid number
- Then, try to fill in the next empty cell in the same way. If an empty cell that does not have any valid numbers is reached, backtrack to the most recently filled cell and increment its number.
- If the number cannot be incremented, then we backtrack again
- Continue doing this until there are no more empty cells on the board. This means,
  - Solution is found, or
  - Backtracked to the first unfilled cell. In this case, no solution exists for the board
- Uses depth first search approach



Back Tracking Algorithm

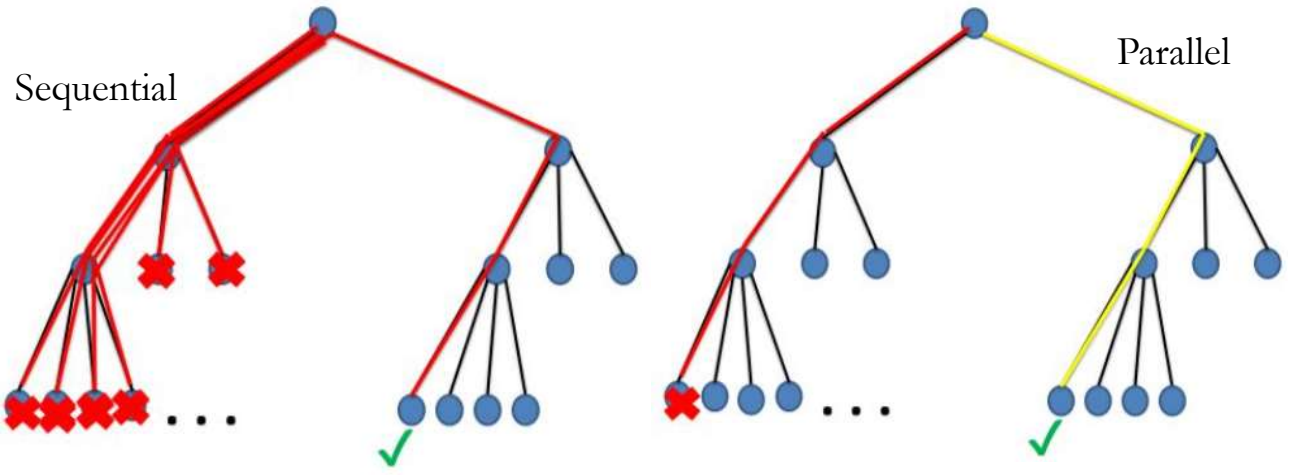


Back Tracking Algorithm



Parallel Approaches

- Working on each cell/row/column/block in parallel is difficult:
  - Work done in thread is dependent on threads or must wait till other threads finish the work
  - Hard to determine what was the previous step and backtrack to it if multiple threads are doing same task
- Approach Chosen:
  - Create a board for each thread with all permutations of valid numbers of the first few empty cells.
  - Use the boards created as an input for each thread
- Example of this approach is shown below:



Challenges

- Multiple threads trying to solve, and first possible solution needs to be used.
- Share data between threads
- Threads needs to be cancelled/exited before they complete if solution is already found by other thread.

```
// Structure for thread arguments
typedef struct {
    int index;
    int tempGrid[SIZE][SIZE];    /*Input Grid for Thread*/
    int row;                     /* to store current empty cell row number*/
    int col;                     /* to store current empty cell colomn number*/
    int num;                     /* to store possible solution for cell that thread is solving*/
    int res;                     /* Result of the thread-if solution is found or not*/
} TH_args;
```

- Declaration and Initialization of global variables, mutex and condition variables

```
int puzzle_solved = 0;
int (*solution)[SIZE];
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t cond = PTHREAD_COND_INITIALIZER;
```

Source Code

Assumption: Input puzzle is Valid

- If solution is found, threads will:
  - Lock Mutex
  - Update global variables indicating puzzle solved along with solution
  - Issue Signal condition that puzzle is solved

```
pthread_mutex_lock(&mutex);
puzzle_solved = 1;
sol = (*args_ptr).index;
solution = (*args_ptr).tempGrid;
pthread_cond_signal(&cond);
pthread_mutex_unlock(&mutex);
```

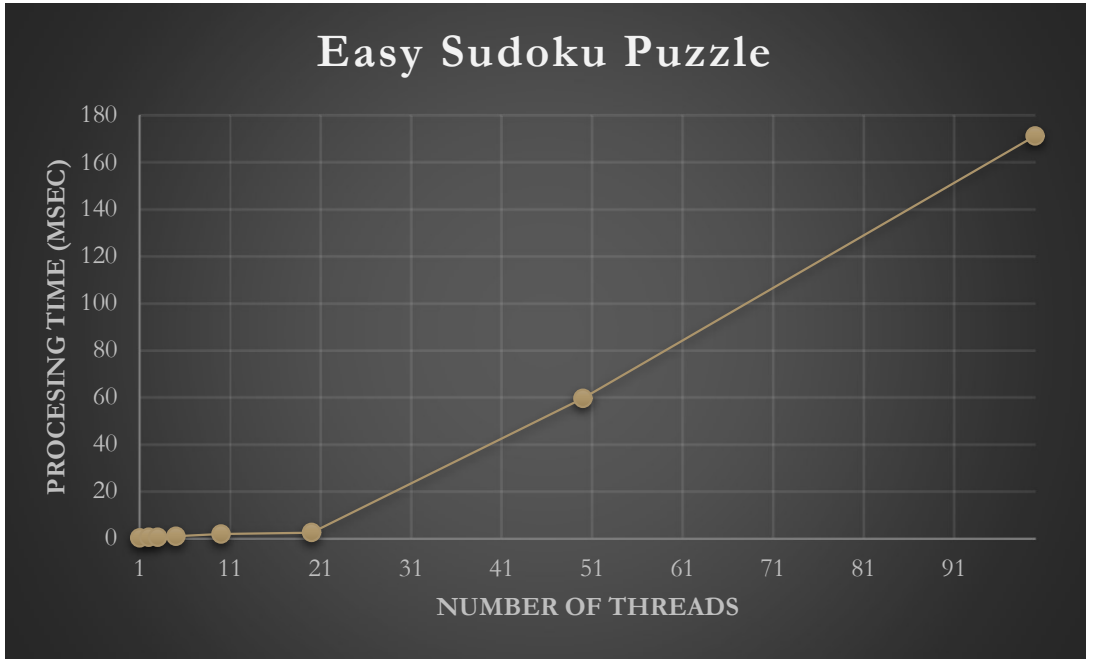
- Main will:
  - Lock Mutex
  - Wait until the puzzle is solved by any thread

```
// waiting for solution
pthread_mutex_lock(&mutex);
while(puzzle_solved == 0)
{
    pthread_cond_wait(&cond,&mutex);
}
pthread_mutex_unlock(&mutex);
```

  - Cancel all threads with out waiting to complete (pthread_cancel)
- (Or) All threads can exit if puzzle is solved and main can wait till all threads complete.

Results

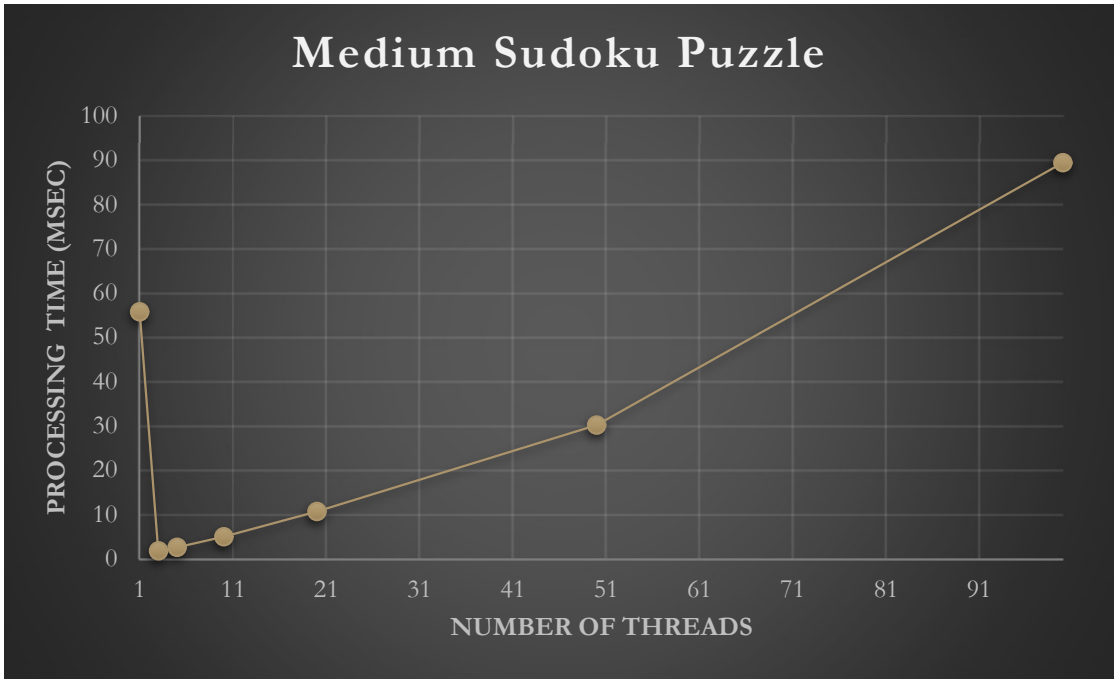
Puzzle Level: Easy



- No Backtracking required.
- Processing time for Sequential execution is 98.6usec and 443usec for Parallel processing with 2 threads.
- Search Path for solution is very less, so sequential approach is better in this case and adding number of threads increases processing time.

Results

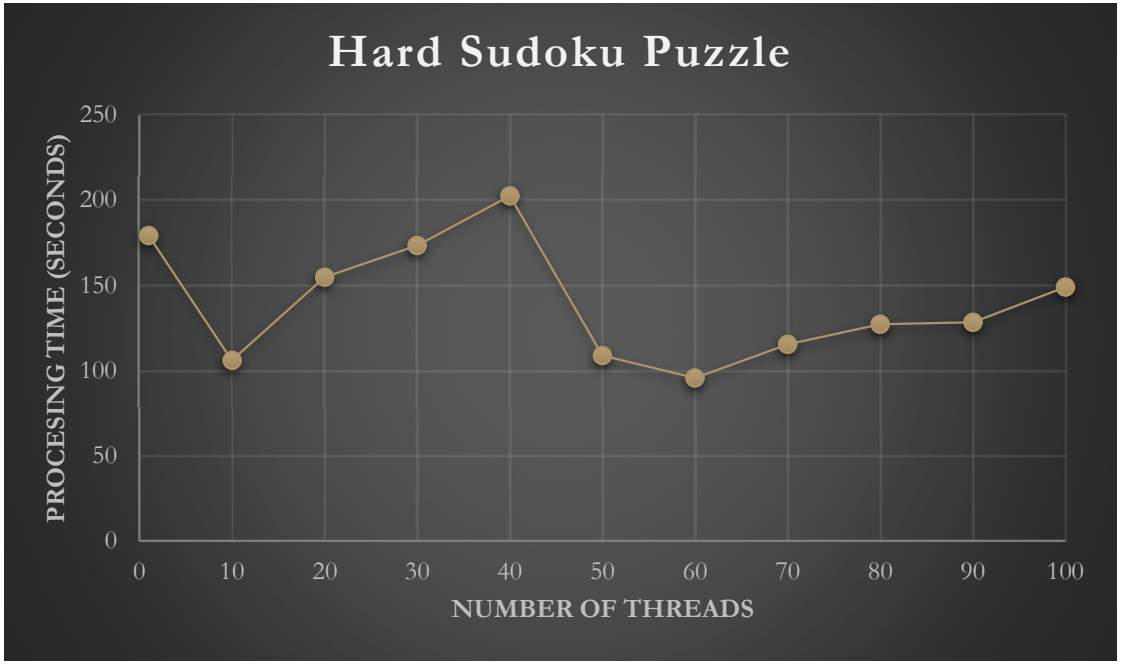
Puzzle Level: Medium



- Minimal Backtracking required.
- Processing time for Sequential execution is 55.7msec and 1.7msec for Parallel processing with 3 threads
- Even after the solution is found, time to find the solution increases:
  - Amount of time each thread gets is less if there are more threads created
  - Overhead in creating threads

Results

Puzzle Level: Extremely Hard



- Input puzzle has lot of empty cells in the beginning and multiple backtrackings required to solve the puzzle
- Sequential approach took around 3 minutes to solve. While parallel approach (best) took ~1.5 minutes
- Execution time reduces if the number of input grids crated is close to the solution.

References

- https://en.wikipedia.org/wiki/Sudoku_solving_algorithms
- https://hackernoon.com/sudoku-and-backtracking-6613d33229af
- Puzzles taken from : https://www.dadsworksheets.com/puzzles/sudoku.html
- https://alitarhini.wordpress.com/2012/02/27/parallel-sudoku-solver-algorithm/
- https://hpc-tutorials.llnl.gov/posix/

Thank You