# Sudoku Validator

**Jami Rithvik**

29.01.2025

## INTRODUCTION

The assignment evaluates a given sudoku and decides whether the sudoku is valid or not in a multithreaded manner.

We evaluated the sudoku in a multithreaded manner using two different algorithms.

## IMPORTANT LIBRARIES USED

1. <pthread.h> : for using threads.
2. <fstream.h>,<sstream.h> : for file input and output.
3. <chrono> : for calculating time taken and time stamp purposes.
4. <atomic> : for avoiding race conditions.

## DESIGN OVERVIEW

The main function takes the input from the file *"inp.txt"* and processes it into a vector of vectors named sudoku(A global variable).

This now is segregated into rows, columns and grids(which are also global variables) using the `segregator()` function.

For the sake of passing a single argument in a thread which is utilized by the function, I created a **class** called `t_data` which contained:

1. The thread number of the utilized thread.
2. The first row/column/grid number which is evaluated(as we use a single thread to verify multiple rows/column/grids)
3. An integer step which is utilized in the mixed method to determine the jump.
4. A boolean to determine if the given row/column/grid is valid or not.
5. Lastly, the chunk size in case when using the chunk.

- I also defined 2 different constructors for use: (for chunk and mixed).

- I used atomic bool to verify whether the whole sudoku is valid or not to avoid race conditions.

### Now let's go into the different types of algorithms printed:

1.**Chunk method:**

- This method was executed by the `exec_chunk()` function.
- It also takes a `vector` of `t_data`.
- I divided the total no.of threads into 3 parts for rows,columns and grids.
- I created a *lambda* function to determine how many row/… each thread should evaluate.
- Then we evaluate all the rows,columns and grids chunk wise by creating threads and calling the `void *` functions created beforehand to verify whether the given row/… is valid or not.
- I added a timestamp at the beginning of the function and at the ending to calculate duration of the function for the sake of comparison.
- If any one of them is invalid then the sudoku is invalid.

2.**Mixed method:**

- This method was executed by the `exec_chunk()` function.
- It also takes a `vector` of `t_data`.
- I divided the total no.of threads into 3 parts for rows,columns and grids.
- We evaluate every alternate row(using `step` variable in `t_data` to jump) and evaluate with the designated thread.
- I added a timestamp at the beginning of the function and at the ending to calculate duration of the function for the sake of comparison.
- If any one of them is invalid then the sudoku is invalid.

- I implemented buffers for each thread and algorithm(mixed and chunk) to avoid data racing during printing the statements since we did not use any other libraries to ensure synchronisation of threads such as MUTEX.

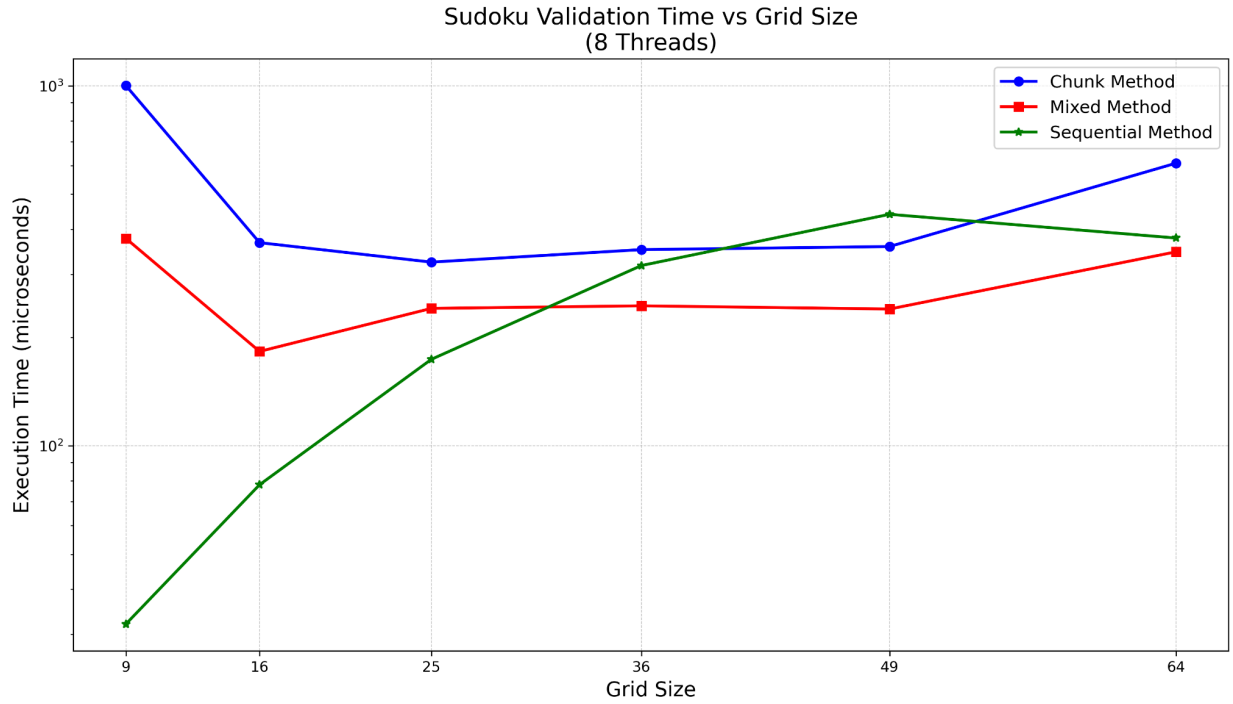
3.**Sequential method:**

- This is the algorithm to compute using a single thread.
- I implemented this by simply traversing through all rows,columns, and grids and verifying whether each of them is valid or not.

# EXPERIMENTS :

## EXPERIMENT 1

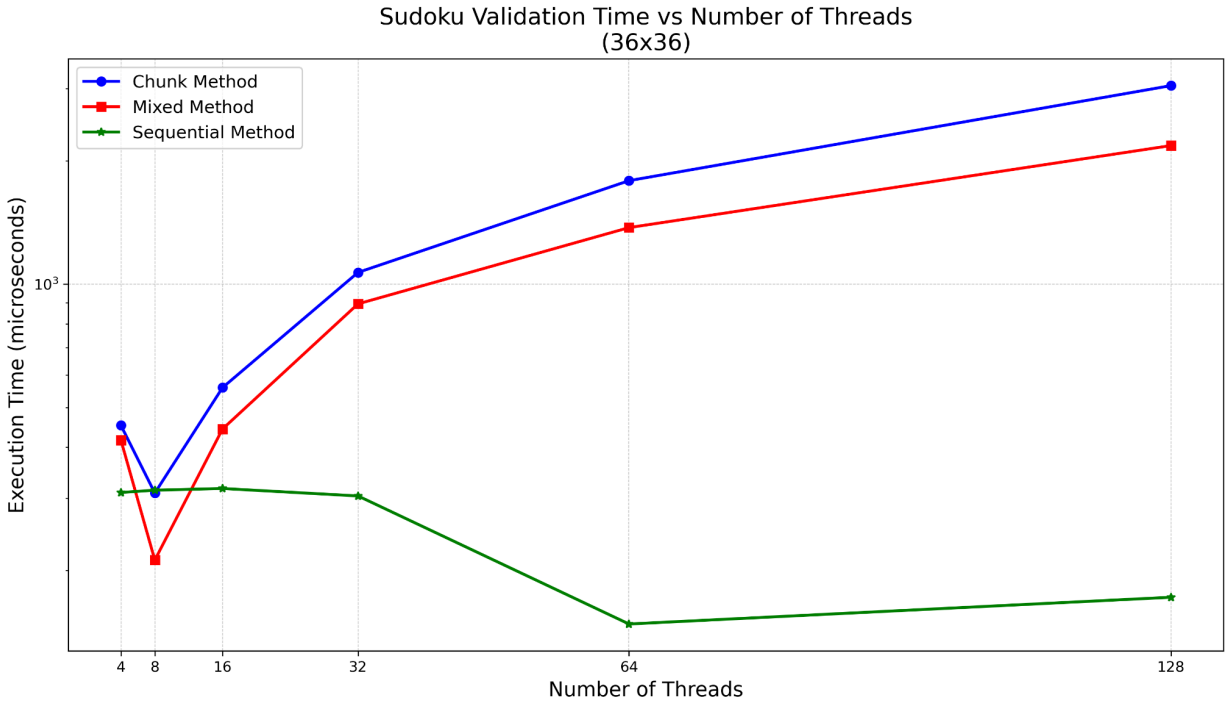
Lets compare the time taken according to the grid size keeping threads constant.




- We can observe that the execution time is low for sequential execution for low input but constantly increases with the size of the grid.
- The discrepancies in our expected time taken may be due to internal processes.
- Initially it is lower than the chunk and mixed methods Because of the thread creation overhead in the multithreaded processes.
- We can also see that mixed is faster than chunk for almost all sizes of grids.
- We can say that mixed is faster than chunk due to better utilization of cache line prefetching (with regards to spatial locality).

## EXPERIMENT 2:

Now, we will compare how varying the no.of threads changes the time taken to evaluate.




- By changing the no.of threads time taken by the sequential process should remain unchanged(but due to internal processes of the CPU there are some minor changes).
- And,for multi threaded processes the execution time will decrease for increase in no.of thread which are equal to the numbers of cores in our CPU but after that the execution times increases due to sharing of the CPU of utilization for multiple threads.
- And as argued above,mixed execution will be  faster than chunk execution.

## CONCLUSION

- For small sized grids sequential processes are faster because multi threaded processes have thread creation overheads.
- Mixed algorithm is more efficient than Chunk due to more effective usage cache line prefetching.