REPORT OF SUDOKU_VALIDATOR

This report presents an implementation of Sudoku validation using three different approaches: Sequential, Chunk, and Mixed methods. The program validates Sudoku puzzles using multi-threading techniques to optimize performance. The goal is to verify whether a given Sudoku solution adheres to the game’s rules while exploring different levels of parallelism for efficiency.

IMPLEMENTATION:

#(Sudoku_methods.c):

In this file ,I have declared some global values:

- N: Dimension of the Sudoku grid (N x N).
- K: Number of threads used for parallel processing.
- sudoku: 2D array storing the Sudoku grid.
- is_sudoku_valid: Boolean flag to track overall validity.
- o_fp: File pointer for writing output logs.
- pthread_mutex_t lock: Mutex for synchronization in multi-threaded execution.

And also defined structure for to store data in thread like starting index,ending index,total no. of threads and thread IDs for thread.

I have done validation conditions for N X N Sudoku,i.e

- checking unique values and bounded between 1 to N for each rows, similarly for each columns.
- checking unique values and bounded between 1 to N for each of n X n subgrids.

For sequential method,
check_row(int **sudoku, int row): Ensures each row contains unique numbers.

- check_num[N + 1]: bool_variable to track which numbers have been seen in the row.
- -for(loop):iterates to check values bounded between 1 to N and duplicate values also marks is_sudoku_valid false if invalid.

check_column(int **sudoku, int col): Ensures each column contains unique numbers.Similarly like row checking.
check_subgrid(int **sudoku, int idx): Validates subgrids based on their index.

- n: dimension of each subgrid ,row_start and col_start are starting point of subgrids.
- check_num[N + 1]: bool_variable to track which numbers have been seen in the row.
- for(loop):iterates through rows of subgrid to check values bounded between 1 to N and duplicate values also marks is_sudoku_valid false if invalid.

For chunk method, check_rows_chunk(void *arg):

- -local_valid:to track local validity(similar to is_sudoku_valid but within function only).
- -outer_for(loop):runs through rows assigned to thread,which keeps tracking of numbers.

- -inner_for(loop):iterates to check values bounded between 1 to N and duplicate values also marks is_sudoku_valid false if invalid.

check_columns_chunk(void *arg):Similarly like row checking.
check_subgrids_chunk(void *arg):

- -n: dimension of each subgrid ,row_start and col_start are starting point of subgrids.
- -check_num[N + 1]: bool_variable to track which numbers have been seen in the row.
- -for(loop):iterates through rows of subgrid to check values bounded between 1 to N and duplicate values also marks is_sudoku_valid false if invalid.

For mixed method,Similar to chunk method's validations but here in this function outermost loop runs till N.
check_rows_mixed(void *arg) check_columns_mixed(void *arg) check_subgrids_mixed(void *arg)

#(Sudoku_Main.c):
In this file, I have created functions for each method(algorithms) so that i can call on my wish which I prefer:

➢ sequential_method(int **sudoku, int N):

- There are three for(loops),they check row, column and subgrid respectively also prints output in output.txt

➢ chunk_method():Dividing the work into chunks assigned to multiple threads

- -chunk_size: to divide the work i.e,row,columns or grids.If (N % K)!=0 then also threads works if any rows is left it is validated by last last thread.
- -Creates arrays of thread_ids,starting_positions and ending_positions,it makes thread to choose chunks.
- -Here thread are created and after finishing task they are joined to parent thread.

➢ mixed_method():Distributing the work across threads in a cyclic manner.

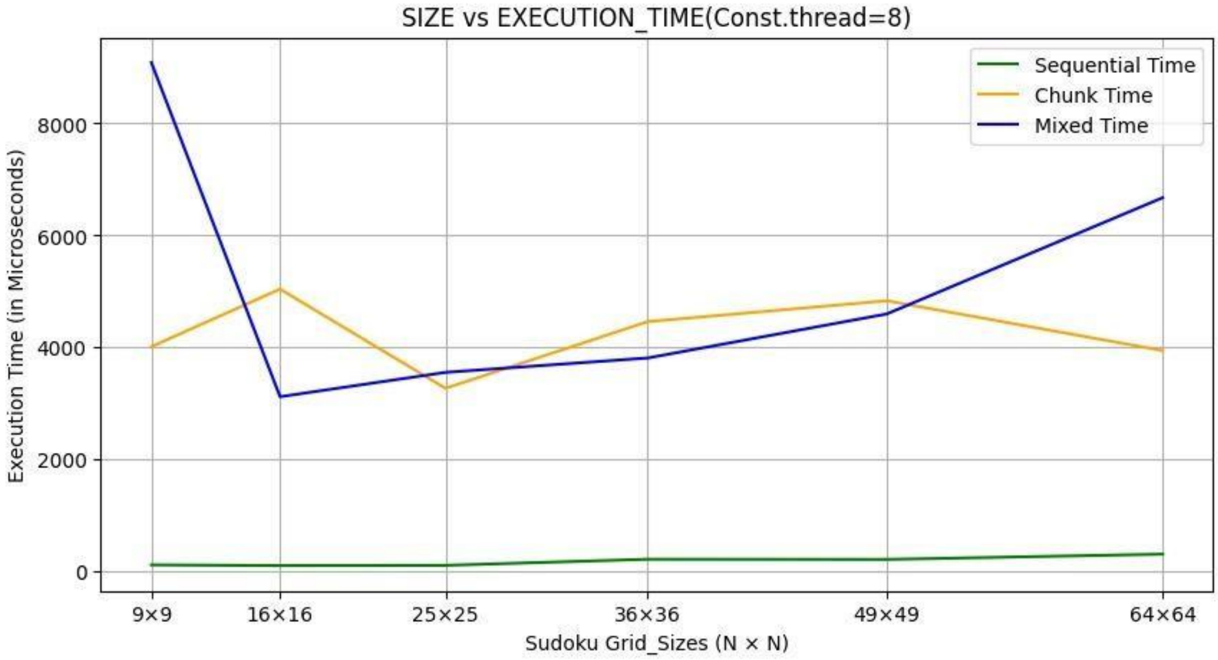
- -Creates arrays of thread_ids and thread_limits.
- -Here thread are created and after finishing task they are joined to parent thread.

#""!!!NOTE: In this methods, I have commented some line of code which are used for ealy termination!!!""

In main() function:

- Reads Sudoku data from inp.txt by opening the file in read_mode.
- input format:
- K N
- <sudoku grid>
- -Allocates memory for sudoku and stores values.
- Asks the user to choose a method (Sequential, Chunk, or Mixed).
- -Executes the selected validation approach.
- Records the validation results in output.txt by opening in write_mode.
- Measures execution time in microseconds and records in output.txt.
- Now closes the input and output file and dealloctes sudoku.

GRAPHS:



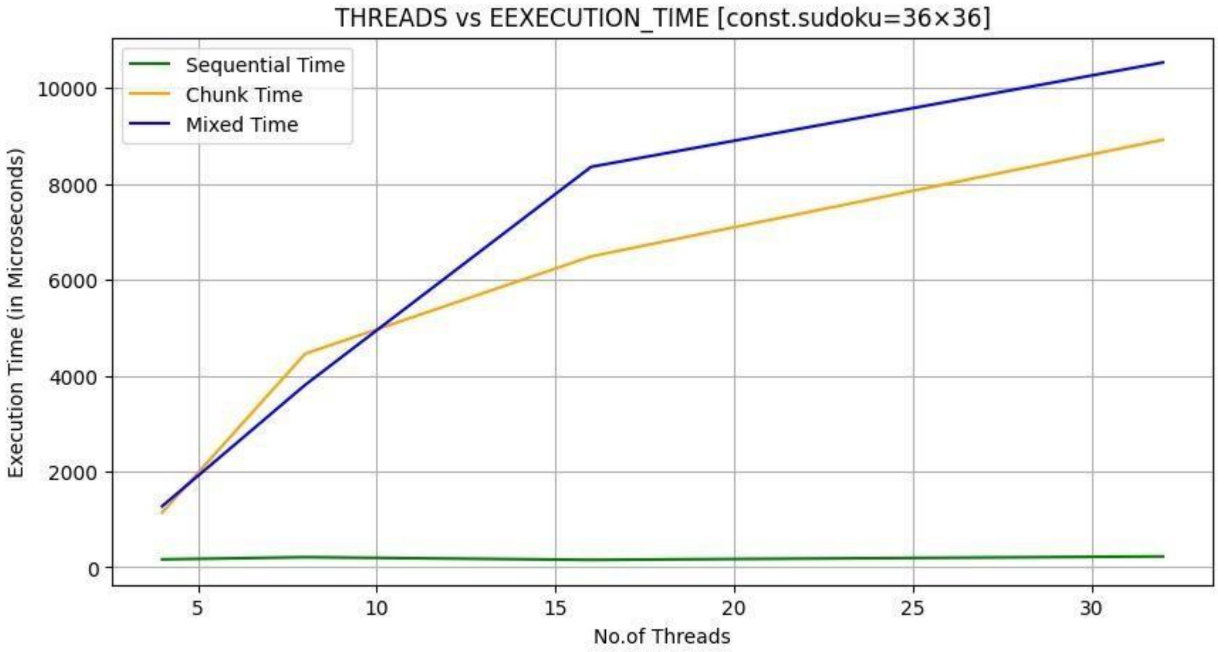
ANALYSIS: To draw this graph:
Experiment-1: In this experiment,I have to kept the number of threads constant say 8 and compare the time taken to validate the sudoku by varying the size as follows: 9X9, 16X16, 25X25, 36X36, 49X49 and 64X64. Thus, in this experiment, the size of the sudoku will be on the x-axis as described above and the y-axis will show the time taken. Then, I have taken an average of 4 execution times of same size of sudoku(Example 16x16 and t1,t2,t3,t4 are execution times then, avg=(t1+t2+t3+t4)/4)

Observations:
1. Sequential Method (Green Line)
o The sequential method exhibits a near-linear growth in execution time as the grid size increases.
o It remains the slowest-growing method
o Even for large grid sizes like 64×64, its execution time remains relatively low compared to the parallel methods.
2. Chunk Method (Orange Line) o The chunk-based approach shows fluctuating performance.
o Initially, it performs better than the mixed method (at 9×9 and 16×16), but as grid size increases, its execution time stabilizes at a higher level.
3. Mixed Method (Blue Line)
o The mixed method exhibits inconsistent performance, with a significant peak at 9×9.
o After an initial drop at 16×16, it shows steady but higher growth compared to the chunk method.
o This suggests that while the mixed approach benefits from parallelism, the additional thread management overhead outweighs the benefits for smaller grids.
Important observations:

- For small grids (9×9, 16×16): The mixed method has the highest execution time, indicating excessive thread synchronization overhead. • For medium grids (25×25 to 49×49): The chunk method performs better than the mixed method, likely due to better task division.
- For large grids (64×64): The mixed method becomes increasingly inefficient compared to the chunk method, indicating that thread overhead and memory contention play a significant role..

Conclusion:

o The sequential method is efficient for small grids but impractical for large ones. o The chunk method provides better performance across most sizes. o The mixed method, while leveraging parallelism, suffers from thread overhead, making it less effective.



Experiment-2:In the second experiment, you have to keep the size of the sudoku constant 36X36 andcompare the performance by varying the number of threads from 4 to 32 (i.e,4, 8, 16 and 32).In this experiment, the number of threads will be on the x-axis and they-axis will show the time taken.Then, I have taken an average of 4 execution times of same size of sudoku(Example 16x16 and t1,t2,t3,t4 are execution times then, avg=(t1+t2+t3+t4)/4)

Observations:

1.Sequential Method (Green Line)

o The sequential method exhibits a constant(nearly) in execution time as threads increase.

2. Chunk Method (Orange Line) o The execution time increases linearly as threads increases.

o Initially, it does’t perform better , after taking no.of threads as 8 it shows better result than mixed method.

3. Mixed Method (Blue Line)

- The execution time increases linearly as threads increases.

Key Observations from the Graph

1. For (4 to 8)threads:The chunk method has the highest execution time likely due to better and faster task division.
2. For (16 to 32)threads:The chunk method has more effieciency compared mixed method but comparably very less though so we might expect mixed
method is ineffient for larger number of threads Conclusion

o The sequential method is efficient for small number of threads but impractical for large ones.

o The chunk method provides better performance across most sizes.As mixed method gives better than chunk for small number of threads only