REPORT

OPERATING SYSTEMS - 2
PROGRAMMING ASSIGNMENT -1

Harsha Vardhan Pulavarthi
CO23BTECH11018

Introduction

As per the given assignment, we need to validate a Sudoku in C.

2 Design Techniques can be used for validating the sudoku in Parallel.

1. Chunk: In the chunk method, a thread is assigned consecutive rows, columns or subgrids to validate. For example, if the number of rows is R and the number of threads for validating rows is K1, the chunk size p is calculated as p=R/K1. Thread 1 will validate rows 1 to p, Thread 2 will validate rows p+1 to 2p, Thread 3 will validate rows 2p+1 to 3p and so on. The same allocation method is applied for columns and subgrids.
2. Mixed: Here each thread, instead of assigning consecutive rows, columns or subgrids, the workload is distributed evenly among the threads in a cyclic manner. For example, if the number of rows is R and the number of threads for validating rows is K1, Thread 1 will validate rows 1, K1+1, 2K1+1,... Thread 2 will validate rows 2, K1+2, 2K1+2,... and so on. This pattern continues for all threads.

Code Design:

1. *Chunk Method for Sudoku Validation Using Multi-threading*

In this implementation, we utilize a chunk-based approach to distribute the workload among multiple threads for validating a Sudoku grid. The process begins by reading the input parameters—k (number of threads), N (size of the Sudoku grid), and the Sudoku grid itself—from the input.txt file. The threads are then distributed almost equally among the tasks of validating rows, columns, and subgrids.

Thread Distribution:

The total number of threads k is divided into three groups:

1. Rows: A subset of threads (K1) is assigned to validate the rows of the Sudoku grid.
2. Columns: Another subset of threads (K2) is assigned to validate the columns.
3. Subgrids: The remaining threads (K3) are assigned to validate the subgrids.

This distribution ensures that the workload is balanced among the threads, optimizing the validation process.

Validation Logic:

Each thread validates its assigned chunk of rows, columns, or subgrids using a checker array. The validation process works as follows:

1. Checker Array Initialization: For each row, column, or subgrid, a binary checker array of size N is initialized with value 1.
2. Marking Used Numbers: As the thread iterates through the assigned chunk, it marks the corresponding entry in the checker array as 0 if the number is found in the Sudoku grid.
3. Validation Check: After processing the chunk, the thread sums up the checker array. If the sum is 0, it indicates that all numbers from 1 to N are present in the chunk, and the chunk is valid. Otherwise, the chunk is marked as invalid, and the overall validity of the Sudoku grid is also marked as

invalid. We also use Mutex (Took help from internet sources to learn about it) to prevent Race conditions.

Chunk Allocation:

The chunk size for each thread is calculated based on the number of threads assigned to a specific task. For example:

- If R is the number of rows and K1 is the number of threads assigned to validate rows, the chunk size p is calculated as p = R / K1.

  Thread 1 validates rows 1 to p.

  Thread 2 validates rows p+1 to 2p.

  Thread 3 validates rows 2p+1 to 3p, and so on.

- The same allocation method is applied for columns and subgrids.

This approach ensures that each thread processes an approximately equal portion of the Sudoku grid, minimizing idle time and maximizing parallelism.

Handling Edge Cases:

If the number of rows, columns, or subgrids is not perfectly divisible by the number of threads, the last thread in each category handles the remaining rows, columns, or subgrids. This ensures that no part of the grid is left unvalidated.

Output:

The program logs the results of each thread's validation to the output.txt file, indicating whether each row, column, or subgrid is valid or invalid. Finally, it provides an overall validity status for the Sudoku grid and the total execution time.

2. *Mixed Method for Sudoku Validation Using Multi-threading*

In this implementation, we utilize a Mixed thread-based approach to distribute the workload among multiple threads for validating a Sudoku grid. The process begins by reading the input parameters—k (number of threads), N (size of the Sudoku grid), and the Sudoku grid itself—from the input.txt file. The threads are then distributed almost equally among the tasks of validating rows, columns, and subgrids.

Thread Distribution:

The total number of threads k is divided into three groups:

1. Rows: A subset of threads (K1) is assigned to validate the rows of the Sudoku grid.
2. Columns: Another subset of threads (K2) is assigned to validate the columns.
3. Subgrids: The remaining threads (K3) are assigned to validate the subgrids.

This distribution ensures that the workload is balanced among the threads, optimizing the validation process.

Validation Logic:

Each thread validates its assigned chunk of rows, columns, or subgrids using a checker array. The validation process works as follows:

1. Checker Array Initialization: For each row, column, or subgrid, a binary checker array of size N is initialized with value 1.
2. Marking Used Numbers: As the thread iterates through the assigned chunk, it marks the corresponding entry in the binary checker array as 0 if the number is found in the Sudoku grid.
3. Validation Check: After processing the chunk, the thread sums up the checker array. If the sum is 0, it indicates that all numbers from 1 to N are present in the chunk, and the chunk is valid. Otherwise, the chunk is marked as invalid, and the overall validity of the Sudoku grid is also marked as invalid. We also use Mutex (Took help of internet sources to learn about it) to prevent Race condition.

Thread Allocation:

We use a cyclic based approach for mixed-threading

- If R is the number of rows and K1 is the number of threads assigned to validate rows, the chunk size p is calculated as p = R / K1.
- In a cyclic pattern, thread 1 validates rows 1, K+1, 2K+1, etc., while thread 2 validates rows 2, K+2, 2K+2, and so forth.
- The same allocation method is applied for columns and subgrids.

  This mixed-threading approach enhances efficiency by ensuring that no thread remains idle while others are working.

Handling Edge Cases:

Here the edge cases are more evenly distributed across the threads unlike Chunk based system, where one thread will get all the extra Rows/Columns/Subgrids.

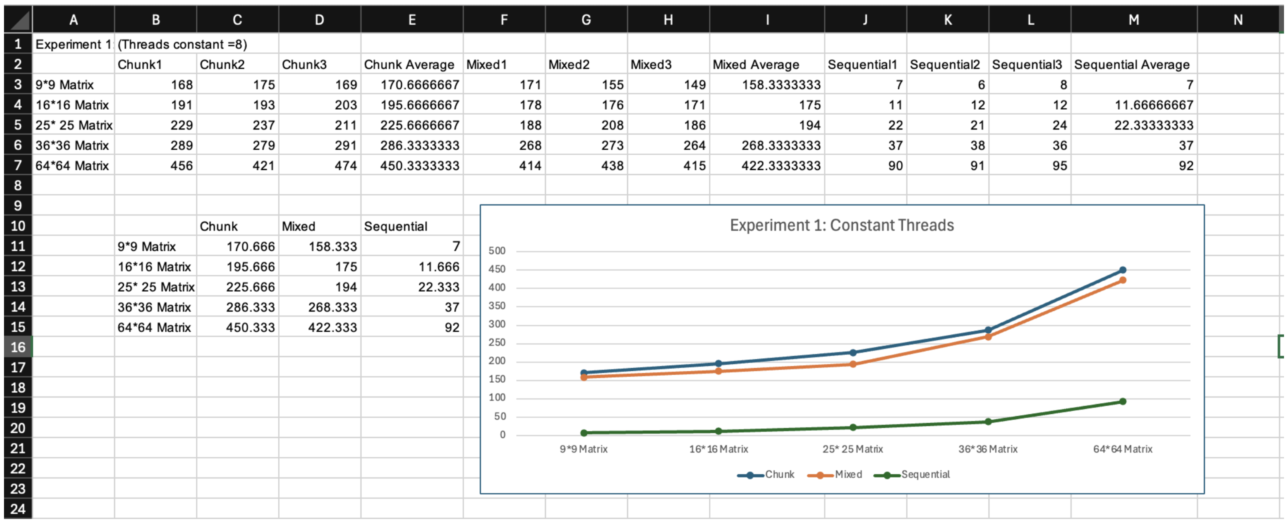
Output:

The program logs the results of each thread's validation to the output.txt file, indicating whether each row, column, or subgrid is valid or invalid. Finally, it provides an overall validity status for the Sudoku grid and the total execution time.

Analysis of Output:

For every test case, we run it 3 times and then take an average of it.

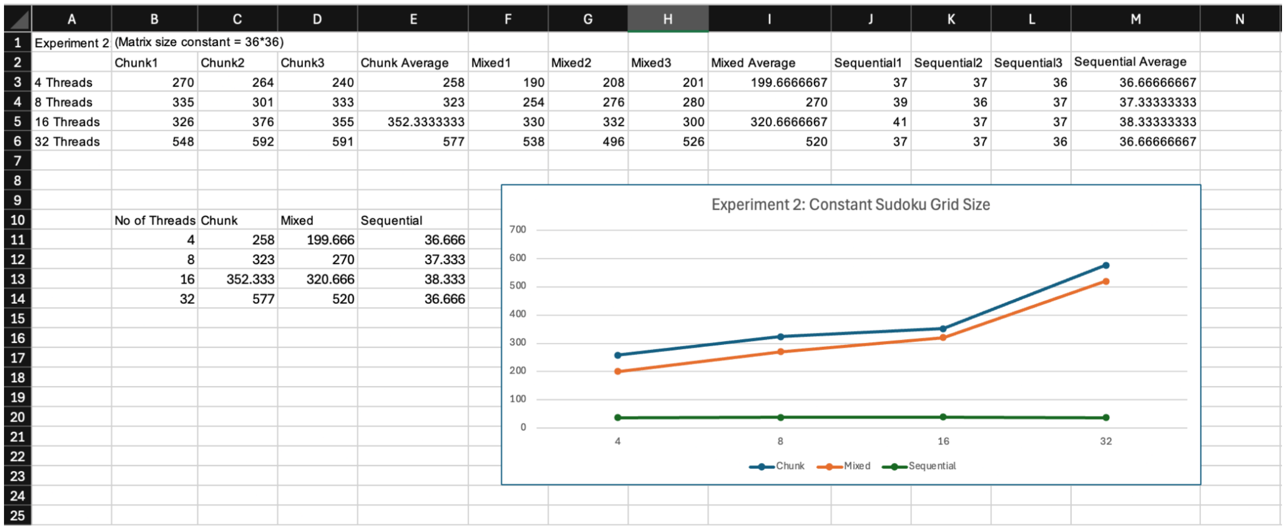
Experiment 1:



Experiment 1	(Threads constant =8)											
	Chunk1	Chunk2	Chunk3	Chunk Average	Mixed1	Mixed2	Mixed3	Mixed Average	Sequential1	Sequential2	Sequential3	Sequential Average
9*9 Matrix	168	175	169	170.6666667	171	155	149	158.3333333	7	6	8	7
16*16 Matrix	191	193	203	195.6666667	178	176	171	175	11	12	12	11.66666667
25* 25 Matrix	229	237	211	225.6666667	188	208	186	194	22	21	24	22.33333333
36*36 Matrix	289	279	291	286.3333333	268	273	264	268.3333333	37	38	36	37
64*64 Matrix	456	421	474	450.3333333	414	438	415	422.3333333	90	91	95	92

	Chunk	Mixed	Sequential
9*9 Matrix	170.666	158.333	7
16*16 Matrix	195.666	175	11.666
25* 25 Matrix	225.666	194	22.333
36*36 Matrix	286.333	268.333	37
64*64 Matrix	450.333	422.333	92

Experiment 2:



Experiment 2	(Matrix size constant = 36*36)											
	Chunk1	Chunk2	Chunk3	Chunk Average	Mixed1	Mixed2	Mixed3	Mixed Average	Sequential1	Sequential2	Sequential3	Sequential Average
4 Threads	270	264	240	258	190	208	201	199.6666667	37	37	36	36.66666667
8 Threads	335	301	333	323	254	276	280	270	39	36	37	37.33333333
16 Threads	326	376	355	352.3333333	330	332	300	320.6666667	41	37	37	38.33333333
32 Threads	548	592	591	577	538	496	526	520	37	37	36	36.66666667

No of Threads	Chunk	Mixed	Sequential
4	258	199.666	36.666
8	323	270	37.333
16	352.333	320.666	38.333
32	577	520	36.666

Analysis:

We notice the following:

1. Sequential Method is the fastest as it does not have the overhead of thread creation and synchronization. Since we have used small sudoku grids, the overhead of multithreading beats its advantages.
2. We notice that as Matrix Size increases, Time taken increases as more computations are done.
3. Overhead of Multithreading also causes an increase in time taken as number of Threads increases (Since we are considering a small Sudoku Grid)
4. The Mixed Method is faster than Chunk Method in all cases possibly due to more balanced divided Rows/Columns/Subgrids among the threads in Mixed Method.

Extra Credit Method:

I have implemented the case of early termination in my chunk Method. If I ever encounter an Invalid Row/Column/Subgrid, it exits the thread and program and directly outputs: Sudoku not Valid.

Note:

I have written similar codes for all 4 functions for ease of checking.

The updated Assignment was shared less than an hour before the deadline, I had written my entire code by then. I had already written the codes using Mutex (learnt it from the internet) and did not include timestamps by then. Mutex can be removed, in most cases there was no race condition, but I added it just incase. I could not add timestamps as well as that changed the run time and was not in line with the data I drew my graphs with.

I have done everything else, including taking multiple results and averaging them for the output. I have attached the exact numbers I have gotten as well.
Do Consider.

Thank You!