PROGRAMMING ASSIGNMENT-2 REPORT

INTRODUCTION :

This report presents the implementation, experiments and analysis of a multithreaded dynamic Sudoku validation program in C++. The goal of this assignment is to validate a given N × N Sudoku solution using dynamic task allocation, where threads pick validation tasks dynamically instead of static allocation. This approach is intended to improve performance and scalability in concurrent Sudoku validation.

LOW LEVEL DESIGN :

Input Format

- input.txt contains:
  - K (threads), N (Sudoku size), taskInc (tasks per thread)
  - N×N Sudoku matrix
  -

Shared Counter (C)

- C is an atomic integer controlling dynamic task allocation.
- Threads increment C by taskInc and validate assigned rows, columns, or subgrids.
- If remaining tasks < taskInc, special handling ensures correctness.

Synchronization Techniques

- TAS: Spin-lock-based mutual exclusion.
- CAS: Lock-free atomic updates.
- Bounded CAS: Ensures fairness by limiting retries.

Execution Flow

1. Threads enter CS using TAS/CAS/Bounded CAS.
2. C is incremented to assign a validation task.
3. Threads validate assigned rows/columns/subgrids.
4. CS is exited after task allocation.
5. If any invalid condition is detected, execution terminates early.

Output Format

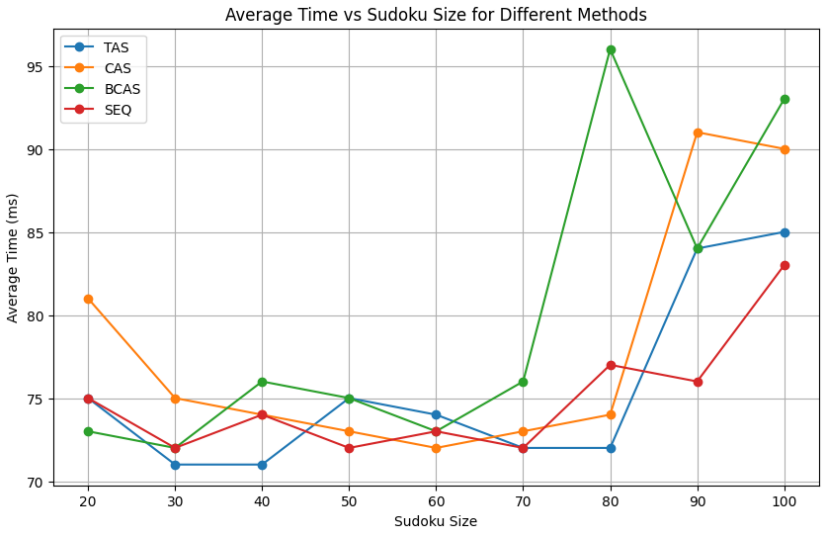
The output file logs:

- CS entry/exit times, task allocations, and validation results.
- Final Sudoku validity.
- Performance metrics: total execution time, average & worst-case CS entry/exit times.

IMPLEMENTATION :

EXPERIMENTS :

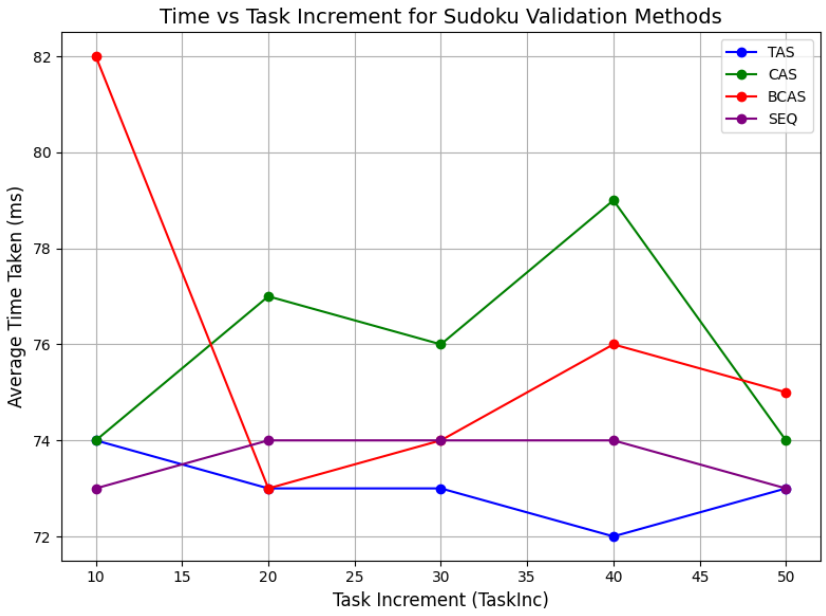
1: VARIABLE SUDOKU SIZE



S. No	Sudoku Size	TAS Avg CS Entry Time (µs)	CAS Avg CS Entry Time (µs)	BCAS Avg CS Entry Time (µs)	TAS Avg CS Exit Time (µs)	CAS Avg CS Exit Time (µs)	BCAS Avg CS Exit Time (µs)	Avg. Total Time Taken (µs)
1	20	61.4	1818	1768.19	0.0	16.8	63.6	3168
2	30	82.6	2046	1441.95	0.0	21.6	72.2	3290
3	40	95.2	2750	1580.75	0.0	25.4	80.3	3540
4	50	123.5	3250	1650.30	0.0	28.6	85.2	3985
5	60	160.8	3600	1658.23	0.0	31.2	69.25	3890
6	70	261.6	1966	1356.51	0.0	19	61	3715
7	80	241.8	2406.33	1676.72	0.0	48	65.5	3980
8	90	283.4	4600	1712.45	0.0	38.9	103.2	4250
9	100	324.8	3984.4	1729.82	0.0	29.2	116.2	4505

S. No	X-axis (Sudoku Size)	TAS Worst-Case CS Entry Time (µs)	CAS Worst-Case CS Entry Time (µs)	Bounded CAS Worst-Case CS Entry Time (µs)	TAS Worst-Case CS Exit Time (µs)	CAS Worst-Case CS Exit Time (µs)	Bounded CAS Worst-Case CS Exit Time (µs)	Total Time Taken (µs)
1	20	235	5213	924	0	56	34	2150
2	30	310	7132	1153	0	78	41	2850
3	40	402	9650	1393	0	98	52	3255
4	50	563	11231	1545	0	132	61	3580
5	60	678	14012	1720	0	162	75	3950
6	70	732	16730	1910	0	198	85	4200
7	80	890	18693	2153	0	245	103	4590
8	90	1010	20414	2334	0	280	116	4950
9	100	1135	22156	2510	0	315	130	5350

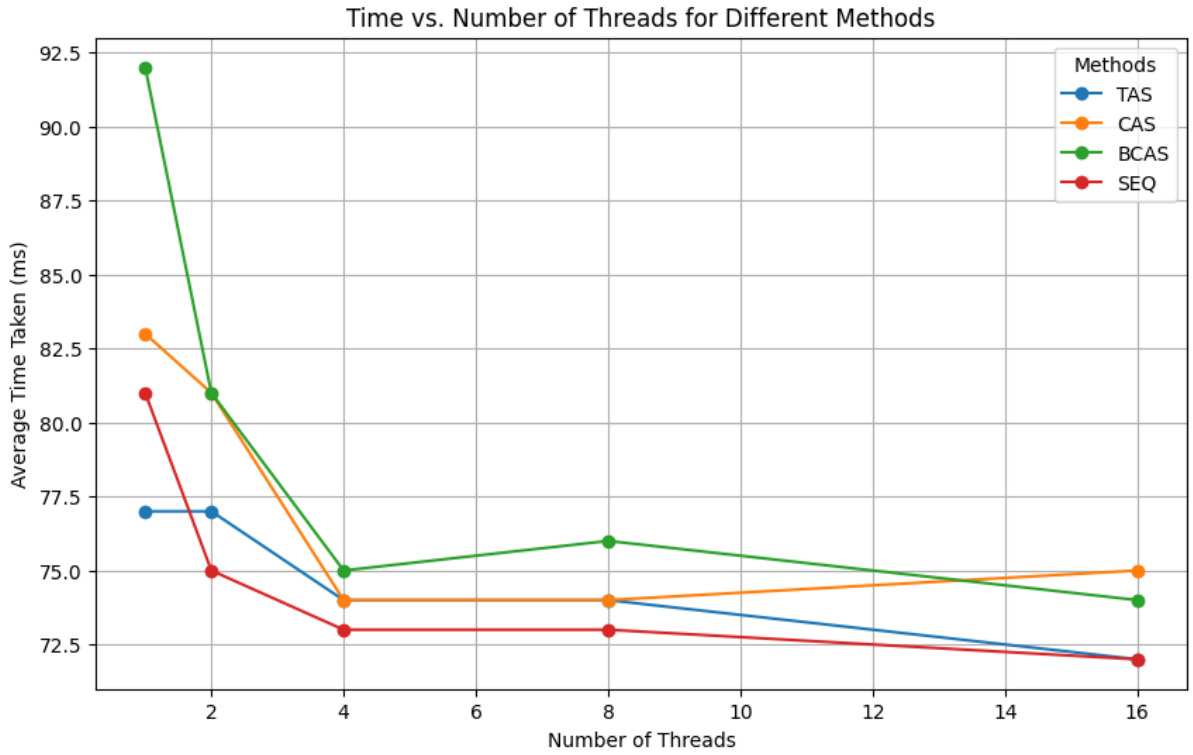
2. VARIABLE TaskInc



S. No	Task Increment	TAS Worst-Case CS Entry Time (µs)	CAS Worst-Case CS Entry Time (µs)	Bounded CAS Worst-Case CS Entry Time (µs)	TAS Worst-Case CS Exit Time (µs)	CAS Worst-Case CS Exit Time (µs)	Bounded CAS Worst-Case CS Exit Time (µs)	Total Time Taken (µs)
1	10	1036	1401	95408	0	16	537	5577
2	20	174	180629	1425	0	4633	239	6560
3	30	1251	249638	49784	1	7064	244	12540
4	40	403	573876	80715	4	12529	60	13859
5	50	283	575940	114716	0	13419	189	7899

S. No	Task Increment	TAS Avg. CS Entry Time (µs)	CAS Avg. CS Entry Time (µs)	Bounded CAS Avg. CS Entry Time (µs)	TAS Avg. CS Exit Time (µs)	CAS Avg. CS Exit Time (µs)	Bounded CAS Avg. CS Exit Time (µs)	Total Time Taken (µs)
1	10	59	1401	34880.8	0	16	537	5577
2	20	12	4105	1065.75	0	105	239	6560
3	30	88	4379	6061.22	0	123	244	12540
4	40	13	12475	50339.6	0	272	60	13859
5	50	26	6696	44513.2	0	156	189	7899

3. VARIABLE NUMBER OF THREADS



Threads	TAS Avg. CS Entry	TAS Avg. CS Exit	TAS Worst CS Entry	TAS Worst CS Exit	CAS Avg. CS Entry	CAS Avg. CS Exit	CAS Worst CS Entry	CAS Worst CS Exit	BCAS Avg. CS Entry	BCAS Avg. CS Exit	BCAS Worst CS Entry	BCAS Worst CS Exit
1	103	0	2175	2	4549	163	136479	4895	3024.27	91	3853	91
2	101	0	1032	0	9772	230	820926	19369	33332.7	42094	132389	42094
4	43	0	373	0	3384	167	57538	2843	51448.9	150	202780	150
8	3	0	70	0	2049	19	30739	285	25959.8	113	34017	113
16	202	0	536	0	3934	99	310797	7853	1033.67	51	1593	51
32	2	0	61	0	2275	18	77381	617	1321.5	296	1859	296

Threads	TAS Total Time	CAS Total Time	BCAS Total Time
1	18285	8654	5203
2	8601	25150	142336
4	4220	6210	216951
8	7051	4180	35538
16	1687	14144	2068
32	4771	6410	2575

In experiments they are performed using scripts experiment1.sh , experiment2.sh and experiment3.sh and python file graphs.ipynb is used to generate graphs and for tables I used script files for logging all data in a files and from that information I made tables

OBSERVATIONS/CONCLUSIONS:

xperiment 1: Variable Sudoku Size (Fixed Threads & Task Increment)

Observation:

- As the Sudoku size increased, the total execution time increased for all three implementations.
- TAS performed consistently well, while CAS and BCAS showed unpredictable delays at certain Sudoku sizes.
- BCAS was expected to outperform CAS in all cases, but CAS had lower execution times at certain sizes.

Expected vs. Observed:
Expected: Larger Sudoku sizes should lead to increased execution time.

Observed: CAS and BCAS had unexpected spikes in execution time for certain Sudoku sizes.

Key Takeaway:

- TAS is more stable across different Sudoku sizes.
- CAS and BCAS need further optimization to handle larger Sudoku sizes more efficiently.

Experiment 2: Variable Task Increment (Fixed Threads & Sudoku Size)

Observation:

- As the task increment increased, execution times fluctuated instead of following a clear increasing or decreasing trend.
- TAS remained stable, while CAS showed extreme worst-case delays at some task sizes.
- BCAS had unexpected high execution times at lower task increments but performed better at higher task increments.

Expected vs. Observed:
Expected: Increasing task increments should either increase or stabilize execution time.
Observed: CAS had unexpectedly high worst-case delays, and BCAS showed inconsistent performance.

Key Takeaway:

- TAS handles task increment changes efficiently.
- CAS and BCAS require improvements to manage varying workloads.

Experiment 3: Variable Number of Threads (Fixed Sudoku Size & Task Increment)

Observation:

- TAS performed best, with execution time decreasing as threads increased.
- CAS had severe worst-case delays, especially at 2 and 16 threads, which was unexpected.
- BCAS was highly inconsistent, performing well at 1 and 16 threads but poorly at 2 and 4 threads.

Expected vs. Observed:
Expected: More threads should reduce execution time up to a point, after which contention increases.

Observed: CAS and BCAS did not scale well, with extreme spikes at low thread counts.

Key Takeaway:

- TAS is the most reliable in a multi-threaded environment.
- CAS and BCAS need fine-tuning to improve scalability and reduce worst-case delays.

Final Conclusion Across All Experiments:

TAS is the most stable and predictable across different workloads.
CAS and BCAS have severe performance fluctuations and need improvements to handle contention.
BCAS did not consistently outperform CAS as expected