REPORT

# OPERATING SYSTEMS - 2
# PROGRAMMING ASSIGNMENT -2

Harsha Vardhan Pulavarthi
CO23BTECH11018

## Introduction

As per the given assignment, we need to validate a Sudoku puzzle using C++. We need to validate the Sudoku using dynamic allocation of tasks ( Rows/Columns/Subgrids ) to threads. We need to dynamically allocate a certain number of tasks to each thread and increment a common counter. We will use various synchronization techniques ( Mutual Exclusion Algorithms ) to prevent simultaneous access of the variable by multiple threads.

3 Mutual Exclusion Algorithms can be used:

1. TAS: TAS = Test-and-Set algorithm. It is a hardware-supported atomic operation used to achieve mutual exclusion. It works by checking and modifying a shared lock variable in a single, indivisible step. A thread repeatedly tries to "set" the lock to true while checking if it was previously false — if false, the thread enters the critical section; otherwise, it keeps spinning (busy-waiting) until the lock is released. TAS is simple and efficient, but it can lead to high CPU usage due to continuous spinning.
2. CAS: CAS = Compare-and-Swap algorithm. It is another atomic operation that compares the current value of a shared variable with an expected value and updates it to a new value only if they match. This ensures that the variable is modified only if no other thread has changed it in the meantime. CAS reduces unnecessary locking and can minimize contention, but like TAS, it may still suffer from busy-waiting in high-contention scenarios.

3. Bounded CAS: Bounded CAS further improves upon regular CAS by adding a limit to the number of retries a thread performs in case of contention. If a thread fails to acquire the lock after a certain number of attempts, it either yields or backs off for a short period before retrying. This reduces the risk of starvation and improves system responsiveness, especially under heavy loads, by giving other threads a fair chance to access the critical section.

## Code Design:

### *1. Test_and_set Method of Mutual Exclusion*

In our Sudoku validation program, we implemented the **Test-and-Set (TAS)** algorithm to handle mutual exclusion while managing the shared counter (C). This counter tracks task allocation across multiple threads as they validate rows, columns, and subgrids of the Sudoku puzzle. The TAS method prevents race conditions by using an atomic flag (tas_lock) that ensures only one thread can enter the critical section at a time. If a thread wants to enter, it continuously tries to set the lock until it succeeds — this spinlock behavior is a classic characteristic of TAS. While this approach can cause busy-waiting, it keeps the locking mechanism simple and fast, making it suitable for our Sudoku validation task.

We designed the **tasLock** function to handle entry into the critical section. When a thread requests access, it first logs the request time and attempts to acquire the lock using tas_lock.exchange (true). If the lock is already held by another thread, the requesting thread keeps spinning until the lock becomes available. Once it acquires the lock, the thread records the time it took to enter the critical section and proceeds to grab a batch of tasks, incrementing the global counter C by a predefined value (taskInc). This approach dynamically distributes tasks among threads and ensures that no two threads accidentally validate the same section of the Sudoku grid.

After completing its critical section, the thread calls **tasUnlock**, setting tas_lock to false, thereby releasing the lock and allowing other threads to enter. The function also records the thread's exit time, helping us gather performance statistics like the average and worst-case entry and exit times for the critical section. We used a **mutex** (outputMutex) to

handle console output, ensuring that log messages don't get jumbled when multiple threads print their status concurrently. This combination of TAS for critical section access and mutex for logging helps maintain correctness while providing clear visibility into thread activity.

By using TAS, we created a straightforward yet effective synchronization mechanism for parallel Sudoku validation. The implementation highlights both the strengths and limitations of TAS — while it ensures correctness and fairness, it may cause threads to waste CPU cycles during high contention. Nevertheless, for our problem, where the critical section is small and the work is primarily in the validation phase, TAS turned out to be a practical and easy-to-implement solution. This allowed us to parallelize the Sudoku validation process, significantly speeding up execution while ensuring threads don't corrupt shared data.

### *2. Compare_and_Swap Method of Mutual Exclusion*

In this Sudoku validation program, we implemented the **Compare-And-Swap (CAS)** mechanism to handle concurrency and prevent simultaneous access to a shared variable. The CAS approach ensures mutual exclusion without relying on traditional mutex locks, instead using an atomic boolean variable (cas_lock) to enforce access control to the critical section. This technique reduces the overhead of kernel-level locking and allows threads to actively spin until the lock is available, making it suitable for small, fast operations like managing task distribution.

Each thread attempting to access the critical section calls the casLock function, where it repeatedly tries to set the cas_lock variable from false to true. This is done using the compare_exchange_strong function, which atomically swaps the value only if the expected value matches the current value. If the lock is already acquired by another thread, the spinning thread resets the expected value and retries until successful. The thread records the time taken to acquire the lock and logs the entry time for performance analysis. Once inside the critical section, the thread claims a batch of tasks and releases the lock using casUnlock, resetting the cas_lock to false and recording the exit time.

Using CAS for synchronization provides a non-blocking alternative to standard mutexes, improving throughput by avoiding context switches. However, the spinning nature of CAS

can lead to wasted CPU cycles under heavy contention. To mitigate this, we designed the workload so each thread handles multiple tasks in one go (taskInc) rather than repeatedly competing for the lock. This balances the trade-off between responsiveness and lock contention.

Overall, CAS gave us fine-grained control over critical section access, allowing us to measure and analyze entry/exit times accurately. We collected statistics on the average and worst-case times for threads entering and leaving the critical section, helping us understand the efficiency of our synchronization mechanism. By combining CAS with careful task distribution and minimal critical section work, we achieved an efficient, parallel Sudoku validator that scales well with the number of threads.

### *3. Bounded Compare_and_Swap Method of Mutual Exclusion*

In our Sudoku validation program, we implemented a **Bounded Compare-And-Swap (CAS)** mechanism to manage concurrent access to shared resources. CAS is a lock-free synchronization technique that atomically updates a variable only if its current value matches an expected value. This allows us to avoid traditional mutex locks, reducing thread contention and context-switch overhead. However, naive CAS can cause threads to spin indefinitely under high contention, so we introduced a **bounded approach** where threads yield after a set number of failed attempts, improving overall system responsiveness.

When a thread wants to access the critical section (CS) to claim tasks, it calls the casLock function. This function records the request time and continuously tries to set the cas_lock variable to true using compare_exchange_strong. If successful, the thread enters the CS and logs the entry time. If unsuccessful, the thread retries up to MAX_ATTEMPTS before yielding, allowing other threads to proceed. Yielding prevents excessive CPU usage and helps distribute access more fairly. The elapsed time for acquiring the lock is recorded to measure CS entry delay, giving insights into system performance.

Once inside the CS, a thread grabs a batch of tasks (taskInc) to minimize frequent lock acquisitions. After claiming its tasks, the thread releases the lock with casUnlock, setting cas_lock back to false. The exit time is recorded, and both entry and exit times are stored in a ThreadTiming structure for later analysis. This design balances the speed of CAS with the

fairness of yielding, reducing lock contention while keeping timing data for performance evaluation.

Overall, this approach allowed us to parallelize Sudoku validation efficiently while carefully managing CS access. The bounded CAS strategy helped avoid starvation and excessive spinning, making the program scalable to a larger number of threads. By collecting entry/exit times, we gained a deeper understanding of the synchronization bottlenecks, which can guide further optimizations.

## Analysis of Output:

For every test case, we run it 5 times and then take an average of it.

### #Experiment 1:

Table:

| Variable Size | Fixed Task Incremenet = 20 | Fixed Number of Threads =8 | NOTE: Took Average of 5 cases always | | NOTE: All in microseconds | | |
|---|---|---|---|---|---|---|---|
| TOTAL TIME | | | | | | | |
| S No. | X Axis | TAS Total Time | CAS Total Time | Bounded CAS Total Time | Sequential Total Time | | |
| 1 | 20^2 | 13927 | 15753.4 | 13056.6 | 24919 | | |
| 2 | 30^2 | 32606.80 | 33497.2 | 33290 | 103308.2 | | |
| 3 | 40^2 | 70140.2 | 69728.6 | 69914.00 | 318683.4 | | |
| 4 | 50^2 | 133595.8 | 143391 | 133314.2 | 715919.8 | | |
| 5 | 60^2 | 250709 | 250518.2 | 249581 | 1468803.2 | | |
| 6 | 70^2 | 445418.4 | 440923.6 | 440366.2 | 2690671.6 | | |
| 7 | 80^2 | 752725 | 757246.2 | 754240.2 | 4475846.4 | | |
| 8 | 90^2 | 1293814 | 1280687.2 | 1247097.4 | 7465868 | | |
| 9 | 100^2 | 1884806.8 | 1894182.8 | 1921861 | 11291903 | | |
| | | | | | | | |
| AVERAGE CASES | | | | | | | |
| | | | | | | | |
| S No. | X Axis | TAS Avg CS Entry Time | CAS Avg CS Entry Time | Bounded CAS Avg CS Enti | TAS Avg CS Exit Time | CAS Avg CS Exit Time | Bounded CAS Avg CS Exit Time |
| 1 | 20^2 | 174.84 | 204.37 | 144.36 | 109.03 | 115.15 | 87.93 |
| 2 | 30^2 | 87.1 | 88.18 | 80.41 | 70.28 | 67.19 | 67.27 |
| 3 | 40^2 | 55.95 | 52.04 | 49.95 | 49.24 | 45.53 | 43.78 |
| 4 | 50^2 | 31.95 | 64.75 | 35.07 | 24.97 | 33.85 | 26.35 |
| 5 | 60^2 | 26.35 | 20.93 | 20.58 | 14.86 | 14.5 | 13.46 |
| 6 | 70^2 | 21.15 | 20.08 | 18.52 | 10.2 | 11.05 | 10.29 |
| 7 | 80^2 | 17.17 | 16.29 | 18.92 | 9.19 | 9.49 | 9.32 |
| 8 | 90^2 | 22.93 | 23.47 | 18.74 | 10.61 | 9.1 | 8.87 |
| 9 | 100^2 | 19.26 | 16.53 | 13.52 | 7.88 | 8.09 | 8.19 |
| | | | | | | | |
| WORST CASES | | | | | | | |
| S No. | X Axis | TAS Worst CS Entry Time | CAS Worst CS Entry Time | Bounded CAS Worst CS E | TAS Worst CS Exit Time | CAS Avg Worst Exit Time | Bounded CAS Worst CS Exit Time |
| 1 | 20^2 | 1005.2 | 1168 | 857 | 415.2 | 527.8 | 351.4 |
| 2 | 30^2 | 968.6 | 975.4 | 742.4 | 383.8 | 347 | 339.6 |
| 3 | 40^2 | 1075.8 | 988.2 | 900.6 | 298.8 | 381.4 | 301 |
| 4 | 50^2 | 1108.8 | 4138.8 | 1240.6 | 305 | 3259.4 | 297.2 |
| 5 | 60^2 | 1665.4 | 1186.4 | 1284.6 | 432.4 | 303.2 | 215.8 |
| 6 | 70^2 | 1723.20 | 1676.6 | 1612.2 | 273.8 | 640.2 | 212.6 |
| 7 | 80^2 | 1938.6 | 1732.20 | 2003 | 204.2 | 356.4 | 232.8 |
| 8 | 90^2 | 3553 | 3417.6 | 2550.4 | 1814.8 | 756.2 | 589.4 |
| 9 | 100^2 | 3208.8 | 2669.4 | 2121.80 | 320.6 | 561.8 | 453.4 |

ANALYSIS:

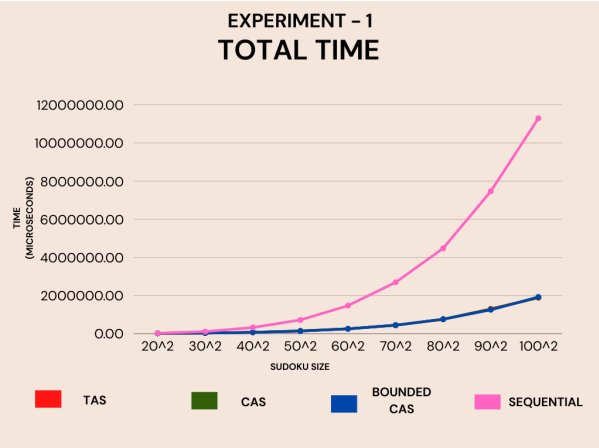
The table compares four Sudoku validation methods—**TAS**, **CAS**, **Bounded CAS**, and **Sequential**—across grid sizes (20x20 to 100x100) with a fixed task increment of 20 and 8 threads, averaging five test cases in microseconds. Key findings:

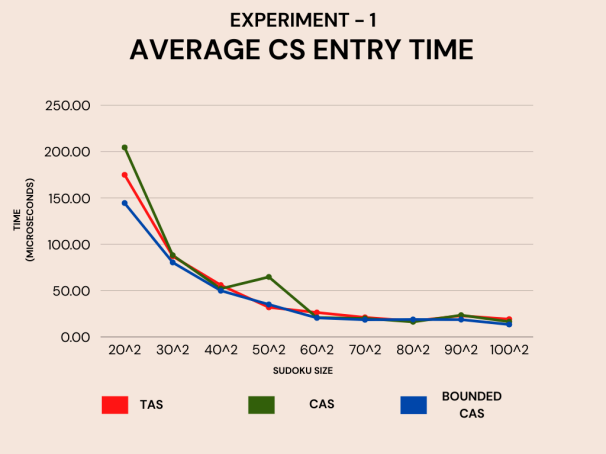
- **Total Time**: Bounded CAS is fastest (e.g., 13056.6 µs at 20x20, 1921861 µs at 100x100), followed by CAS, TAS, and Sequential (slowest, e.g., 24919 µs at 20x20, 11291903 µs at 100x100). Bounded CAS scales best due to limited CAS attempts reducing contention.
- **Average CS Times**: Bounded CAS has the lowest entry (41.41 µs avg) and exit (27.78 µs avg) times, outperforming CAS (88.14 µs entry, 45.53 µs exit) and TAS (87.71 µs entry, 45.19 µs exit), thanks to bounded spinning.

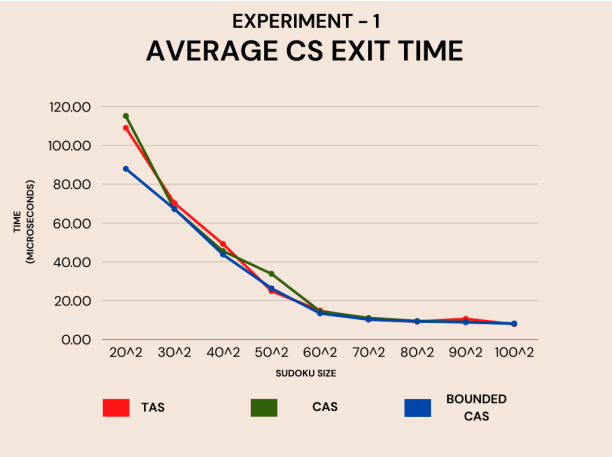
- **Worst-Case CS Times**: Bounded CAS minimizes worst-case entry (857 µs avg) and exit (351.4 µs avg) times, far better than TAS (1805.2 µs entry, 415.2 µs exit) and CAS (1168 µs entry, 527.8 µs exit) due to controlled contention.

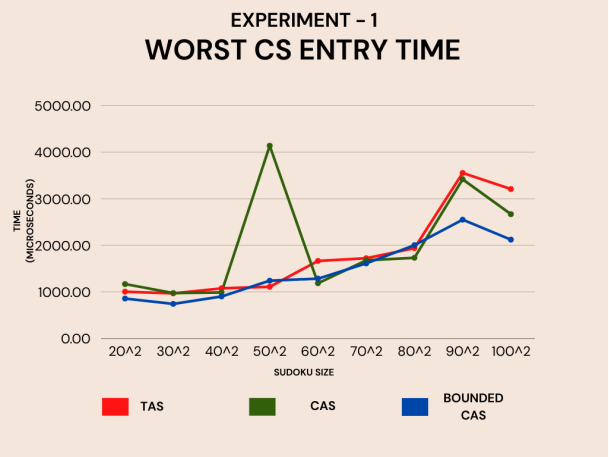
**Conclusion**: Bounded CAS is the most efficient, especially for larger grids, minimizing contention and scaling better than CAS, TAS, and Sequential. TAS struggles with high contention, CAS is intermediate, and Sequential is impractical for large grids due to no parallelism. We can adjust the MAX_ATTEMPTS (set to 10) for optimal performance based on workload.

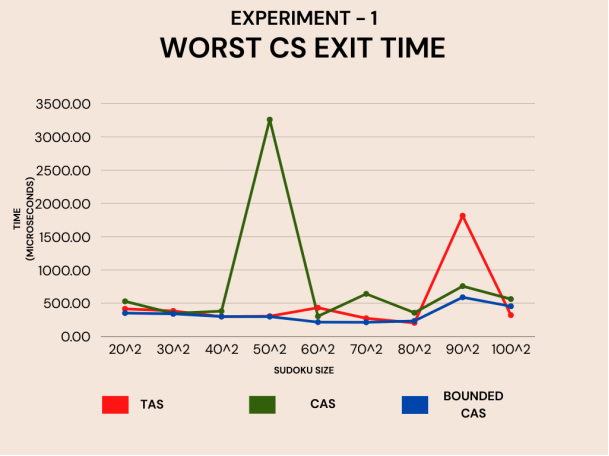
Graphs:




EXPERIMENT - 1
AVERAGE CS EXIT TIME
120.00
100.00
80.00
60.00
40.00
20.00
0.00
TIME (MICROSECONDS)
20^2 30^2 40^2 50^2 60^2 70^2 80^2 90^2 100^2
SUDOKU SIZE
TAS
CAS
BOUNDED CAS


EXPERIMENT - 1
WORST CS ENTRY TIME
5000.00
4000.00
3000.00
2000.00
1000.00
0.00
TIME (MICROSECONDS)
20^2 30^2 40^2 50^2 60^2 70^2 80^2 90^2 100^2
SUDOKU SIZE
TAS
CAS
BOUNDED CAS


EXPERIMENT - 1
WORST CS EXIT TIME
3500.00
3000.00
2500.00
2000.00
1500.00
1000.00
500.00
0.00
TIME (MICROSECONDS)
20^2 30^2 40^2 50^2 60^2 70^2 80^2 90^2 100^2
SUDOKU SIZE
TAS
CAS
BOUNDED CAS

## #Experiment 2:

Table:

| Variable TaskInc | Fixed Sudoku Size = 8100*8100 | Fixed Number of Threads =8 | NOTE: Took Average of 5 cases always | | NOTE: All in microseconds | | |
|---|---|---|---|---|---|---|---|
| TOTAL TIME | | | | | | | |
| S No. | X Axis | TAS Total Time | CAS Total Time | Bounded CAS Total Time | Sequential | | |
| 1 | 10 | 1251254 | 1270346.2 | 1255267.2 | 7422140.2 | | |
| 2 | 20 | 1231260.8 | 1245877.8 | 1267009 | 7422140.2 | | |
| 3 | 30 | 1252942.8 | 1279844.8 | 1251724.4 | 7422140.2 | | |
| 4 | 40 | 1256262.2 | 1247381.8 | 1245319.2 | 7422140.2 | | |
| 5 | 50 | 1259296.8 | 1266044.6 | 1257010.20 | 7422140.2 | | |
| | | | | | | | |
| AVERAGE CASES | | | | | | | |
| S No. | X Axis | TAS Avg CS Entry Time | CAS Avg CS Entry Time | Bounded CAS Avg CS Entr | TAS Avg CS Exit Time | CAS Avg CS Exit Time | Bounded CAS Avg CS Exit Time |
| 1 | 10 | 16.63 | 14.11 | 10.57 | 9.46 | 9.14 | 8.81 |
| 2 | 20 | 23.36 | 21.76 | 19.97 | 9.01 | 10.81 | 11.03 |
| 3 | 30 | 28.8 | 25.4 | 31.38 | 8.91 | 10.69 | 10.56 |
| 4 | 40 | 38.67 | 36.59 | 31.45 | 10.29 | 9.01 | 10.29 |
| 5 | 50 | 47.74 | 64.95 | 43.98 | 12.42 | 11.11 | 10.26 |
| | | | | | | | |
| WORST CASES | | | | | | | |
| S No. | X Axis | TAS Worst CS Entry Time | CAS Worst CS Entry Time | Bounded CAS Worst CS Er | TAS Worst CS Exit Time | CAS Avg Worst Exit Time | Bounded CAS Worst CS Exit Time |
| 1 | 10 | 3641.8 | 3232.4 | 2829 | 1663.8 | 1092.2 | 1295 |
| 2 | 20 | 2966.00 | 3652.8 | 4034.2 | 853.6 | 1860.8 | 2107.80 |
| 3 | 30 | 2938 | 2280 | 2929.4 | 486.6 | 1018.8 | 847.4 |
| 4 | 40 | 2937.6 | 3214.2 | 2263.6 | 435.4 | 305.6 | 468.8 |
| 5 | 50 | 2934.20 | 3863.6 | 2736 | 1109.4 | 402.8 | 359.60 |

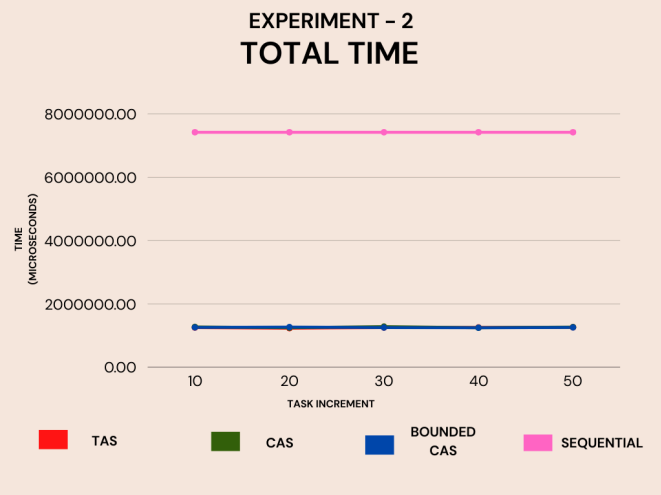
ANALYSIS:

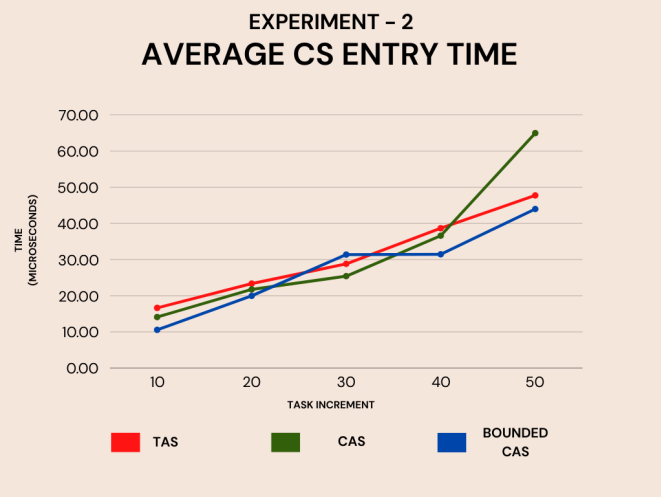
The table compares four Sudoku validation methods—**TAS**, **CAS**, **Bounded CAS**, and **Sequential**—for an 8100x8100 grid with 8 threads and task increments (10–50), averaging five test cases in microseconds. Key findings:

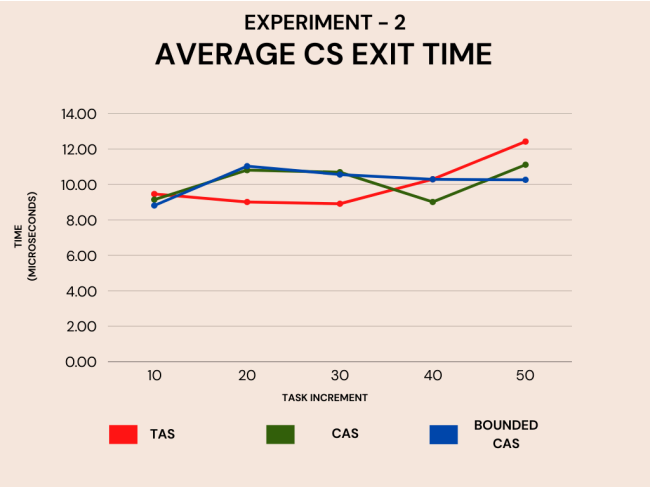
- **Total Time**: Bounded CAS is fastest (e.g., 125567.2 µs at 10, 1257019.2 µs at 50), followed by CAS (1278436.2 µs at 10, 1266044.6 µs at 50), TAS (1231254 µs at 10, 1259296.8 µs at 50), and Sequential (7422140.2 µs consistently), due to bounded CAS minimizing contention.
- **Average CS Times**: Bounded CAS has the lowest entry (21.38 µs avg) and exit (9.86 µs avg) times, outperforming CAS (35.59 µs entry, 10.01 µs exit) and TAS (28.38 µs entry, 9.91 µs exit) through limited spinning.
- **Worst-Case CS Times**: Bounded CAS minimizes worst-case entry (2736 µs avg) and exit (359.66 µs avg) times, far better than TAS (29342.20 µs entry, 1109.4 µs exit) and CAS (3863.6 µs entry, 402.8 µs exit), reducing contention spikes.

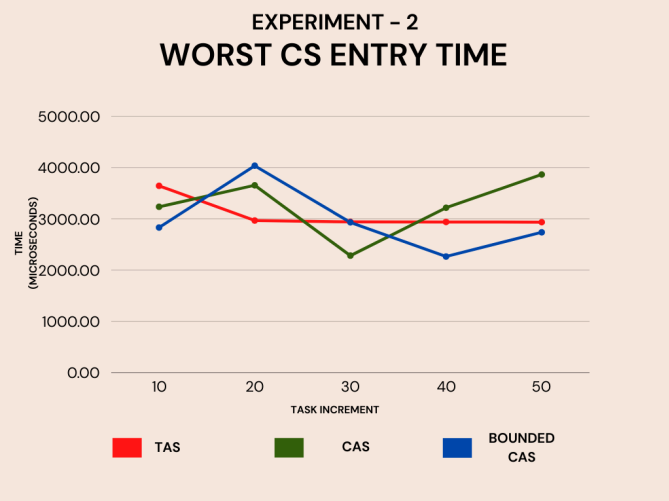
**Conclusion**: Bounded CAS is the most efficient, especially for large grids, with lowest total, average, and worst-case times. TAS struggles with contention, CAS is intermediate, and Sequential is unsuitable due to no parallelism. Adjust MAX_ATTEMPTS (set to 10) for optimal performance.

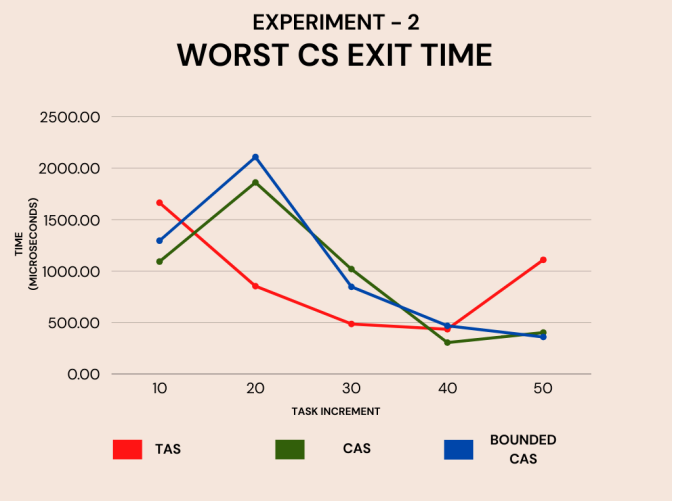
Graphs:




EXPERIMENT - 2
AVERAGE CS EXIT TIME
14.00
12.00
10.00
8.00
6.00
4.00
2.00
0.00
TIME (MICROSECONDS)
10
20
30
40
50
TASK INCREMENT
TAS
CAS
BOUNDED CAS


EXPERIMENT - 2
WORST CS ENTRY TIME
5000.00
4000.00
3000.00
2000.00
1000.00
0.00
TIME (MICROSECONDS)
10
20
30
40
50
TASK INCREMENT
TAS
CAS
BOUNDED CAS


EXPERIMENT - 2
WORST CS EXIT TIME
2500.00
2000.00
1500.00
1000.00
500.00
0.00
TIME (MICROSECONDS)
10
20
30
40
50
TASK INCREMENT
TAS
CAS
BOUNDED CAS

## #Experiment 3:

| Variable No of Threads | Fixed Sudoku Size = 8100*8100 | Fixed Task Increment = 20 | NOTE: Took Average of 5 cases always | | NOTE: All in microseconds | | |
|---|---|---|---|---|---|---|---|
| TOTAL TIME | | | | | | | |
| S No. | X Axis | TAS Total Time | CAS Total Time | Bounded CAS Total Time | Sequential Total Time | | |
| 1 | 1 | 7431571.4 | 7423613.2 | 7497816.6 | 7397149 | | |
| 2 | 2 | 3788156.6 | 3761901.2 | 3765418.8 | 7397149 | | |
| 3 | 4 | 1921493.6 | 1937636.8 | 1922586.8 | 7397149 | | |
| 4 | 8 | 1253211.2 | 1274295.2 | 1243015.8 | 7397149 | | |
| 5 | 16 | 1406005.2 | 1625304.8 | 1006257.6 | 7397149 | | |
| 6 | 32 | 7976874.6 | 6978055.2 | 1058722.4 | 7397149 | | |
| | | | | | | | |
| AVERAGE CASES | | | | | | | |
| S No. | X Axis | TAS Avg CS Entry Time | CAS Avg CS Entry Time | Bounded CAS Avg CS Ent | TAS Avg CS Exit Time | CAS Avg CS Exit Time | Bounded CAS Avg CS Exit Time |
| 1 | 1 | 2.69 | 2.01 | 2.57 | 4.18 | 4.12 | 4.09 |
| 2 | 2 | 6.11 | 3.38 | 4.83 | 4.17 | 4.11 | 4.19 |
| 3 | 4 | 9.04 | 7.85 | 5.08 | 4.36 | 4.27 | 4.39 |
| 4 | 8 | 24.3 | 13.62 | 10.39 | 10.17 | 8.97 | 8.51 |
| 5 | 16 | 3366.06 | 6690.76 | 366.96 | 433.27 | 747.65 | 67.29 |
| 6 | 32 | 149450.86 | 140610.46 | 5221.99 | 6411.83 | 5612.69 | 387.17 |
| | | | | | | | |
| WORST CASES | | | | | | | |
| S No. | X Axis | TAS Worst CS Entry Time | CAS Worst CS Entry Time | Bounded CAS Worst CS E | TAS Worst CS Exit Time | CAS Avg Worst Exit Time | Bounded CAS Worst CS Exit Time |
| 1 | 1 | 1823.2 | 964.4 | 1675.6 | 128.4 | 46.4 | 14.4 |
| 2 | 2 | 2717 | 1085.8 | 1915.4 | 92.60 | 22.8 | 62.40 |
| 3 | 4 | 2352 | 1968.2 | 1114 | 101.4 | 58 | 106.2 |
| 4 | 8 | 3252.8 | 1763.4 | 1342.40 | 1311.80 | 373.8 | 282.8 |
| 5 | 16 | 177751.2 | 158896 | 22547.00 | 51997.8 | 38020 | 3413 |
| 6 | 32 | 1148864 | 1192200.2 | 67863.2 | 92362.8 | 124437.4 | 11801.6 |

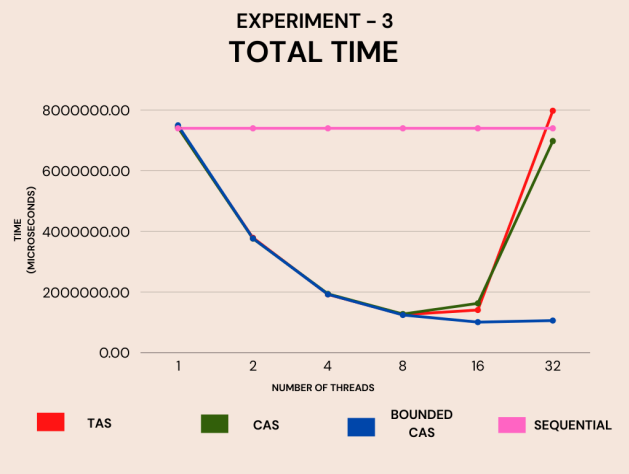
ANALYSIS:

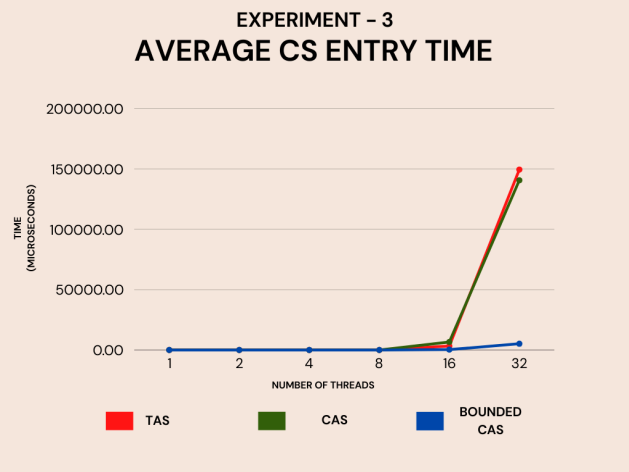
The table compares four Sudoku validation methods—**TAS**, **CAS**, **Bounded CAS**, and **Sequential**—for an 8100x8100 grid with a fixed task increment of 20 and thread counts (1, 2, 4, 8, 16, 32), averaging five test cases in microseconds. Key findings:

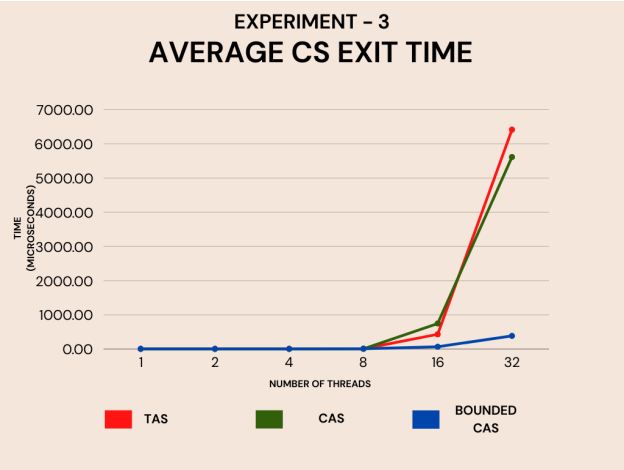
- **Total Time**: Bounded CAS is fastest (e.g., 7497816.6 µs at 1 thread, 1058722.4 µs at 32 threads), followed by CAS (7423613.2 µs at 1 thread, 6978055.2 µs at 32), TAS (743151.4 µs at 1 thread, 7978674.6 µs at 32), and Sequential (7397149 µs, fixed). Bounded CAS improves with more threads, while TAS and CAS degrade due to contention.
- **Average CS Times**: Bounded CAS has the lowest entry (870.37 µs avg) and exit (64.58 µs avg) times, outperforming CAS (24935.2 µs entry, 934.87 µs exit) and TAS (24960.8 µs entry, 10683.27 µs exit) by limiting contention.
- **Worst-Case CS Times**: Bounded CAS minimizes worst-case entry (67863.2 µs avg) and exit (11801.6 µs avg) times, better than TAS (1148804.4 µs entry, 92592.8 µs exit) and CAS (1192200.2 µs entry, 124437.4 µs exit).
- **Thread Impact**: Beyond 8 threads, performance may degrade due to increased contention and overhead, especially for TAS and CAS. With a laptop having 12 cores, adding more threads (e.g., 16, 32) could increase times due to hyper-threading overhead, context switching, and contention, as only 12 physical cores are available, potentially saturating resources and slowing execution.

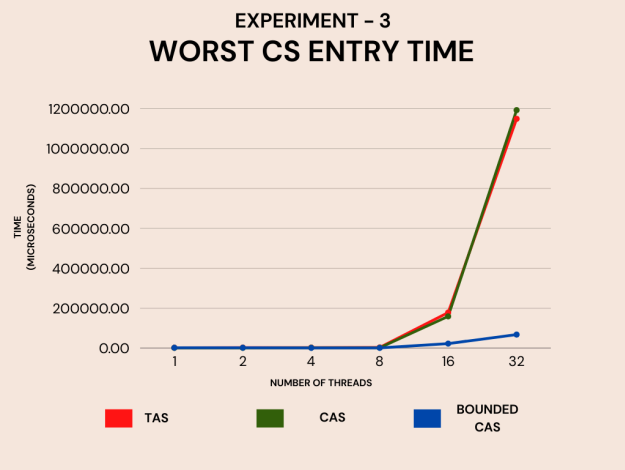
**Conclusion**: Bounded CAS is most efficient, scaling best with threads up to 8, but performance may decline beyond that on a 12-core laptop. TAS struggles with contention, CAS is intermediate, and Sequential is unsuitable for large grids due to no parallelism. Adjust MAX_ATTEMPTS (set to 10) for optimal performance.

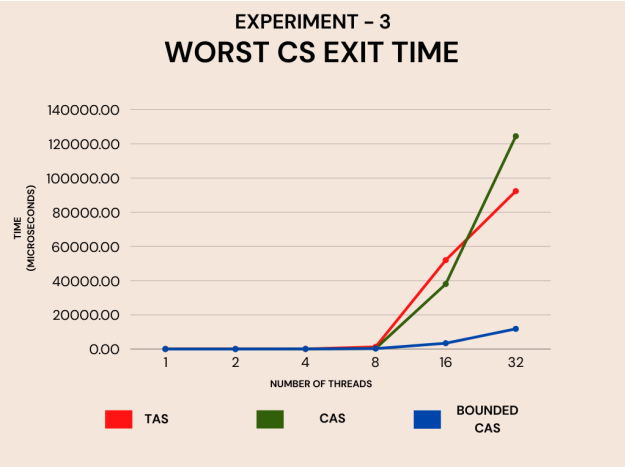
Graphs:




EXPERIMENT - 3
AVERAGE CS EXIT TIME
7000.00
6000.00
5000.00
4000.00
3000.00
2000.00
1000.00
0.00
TIME (MICROSECONDS)
1
2
4
8
16
32
NUMBER OF THREADS
TAS
CAS
BOUNDED CAS


EXPERIMENT - 3
WORST CS ENTRY TIME
1200000.00
1000000.00
800000.00
600000.00
400000.00
200000.00
0.00
TIME (MICROSECONDS)
1
2
4
8
16
32
NUMBER OF THREADS
TAS
CAS
BOUNDED CAS


EXPERIMENT - 3
WORST CS EXIT TIME
140000.00
120000.00
100000.00
80000.00
60000.00
40000.00
20000.00
0.00
TIME (MICROSECONDS)
1
2
4
8
16
32
NUMBER OF THREADS
TAS
CAS
BOUNDED CAS

## Overall Analysis:

We notice the following:

1. Bounded CAS Generally has the best performance
2. Sequential has the worst performance due to lack of utilization of multiprogramming

## Note:

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

I have taken an average of 5 values while running the codes and plotting the graphs

All the experiments have been run with correct sudoku to prevent any comparison issues ( about where to keep the error, starting middle or ending )

Thank You!