Name : Dudekula Dheeraj
Roll No : CS22BTECH11019

Assignment 3

Dynamic Matrix Squaring Report

Objective: Efficiently calculate the square of a matrix by parallelizing the task using threads and dynamic allocation of work.

Algorithms:

Test-and-Set (TAS):

- Uses an atomic flag to manage access to the critical section.
- Only one thread can set the flag (lock) to true at a time.
- Other threads spin-wait until the flag is released.
- Simple but can lead to busy-waiting, where threads continuously check the flag's status.

2. Compare-and-Swap (CAS):

- Uses an atomic integer variable to manage access to the critical section.
- Threads try to swap the value of the lock variable from 0 to 1 atomically.
- If successful, the thread enters the critical section; otherwise, it retries.
- More efficient than TAS as it doesn't involve continuous spinning.

3. Bounded CAS:

- Extension of CAS with an additional vector to manage waiting threads.
- Allows a thread that finishes its work to choose the next waiting thread.
- Ensures fairness by avoiding thread starvation.

4. Increment Atomic (inc_atm):

- Uses an atomic integer variable to divide work among threads.
- Each thread atomically increments a counter to determine its assigned rows.
- Simple and efficient for distributing work evenly among threads.
- Avoids contention for a single lock but requires careful division of work to avoid imbalance.

These algorithms demonstrate different approaches to managing concurrent access to shared resources, each with its trade-offs in terms of simplicity, fairness, and efficiency.

Main Function:

- Reads matrix size (N), number of threads (K), and rows per thread (rowInc) from input file.
- Creates input and output matrices.
- Reads input matrix from file.
- Creates K threads, each executing a specific function (TAS, CAS, bounded_CAS, inc_atm).
- Waits for threads to finish and writes output matrix and execution time to output file.

Conclusion:

Parallelization of matrix squaring using dynamic allocation of work and different mutual exclusion methods improves efficiency compared to a single-threaded approach.

Experiments:

Experiment 1:

Time(sec) vs Size of input (N):

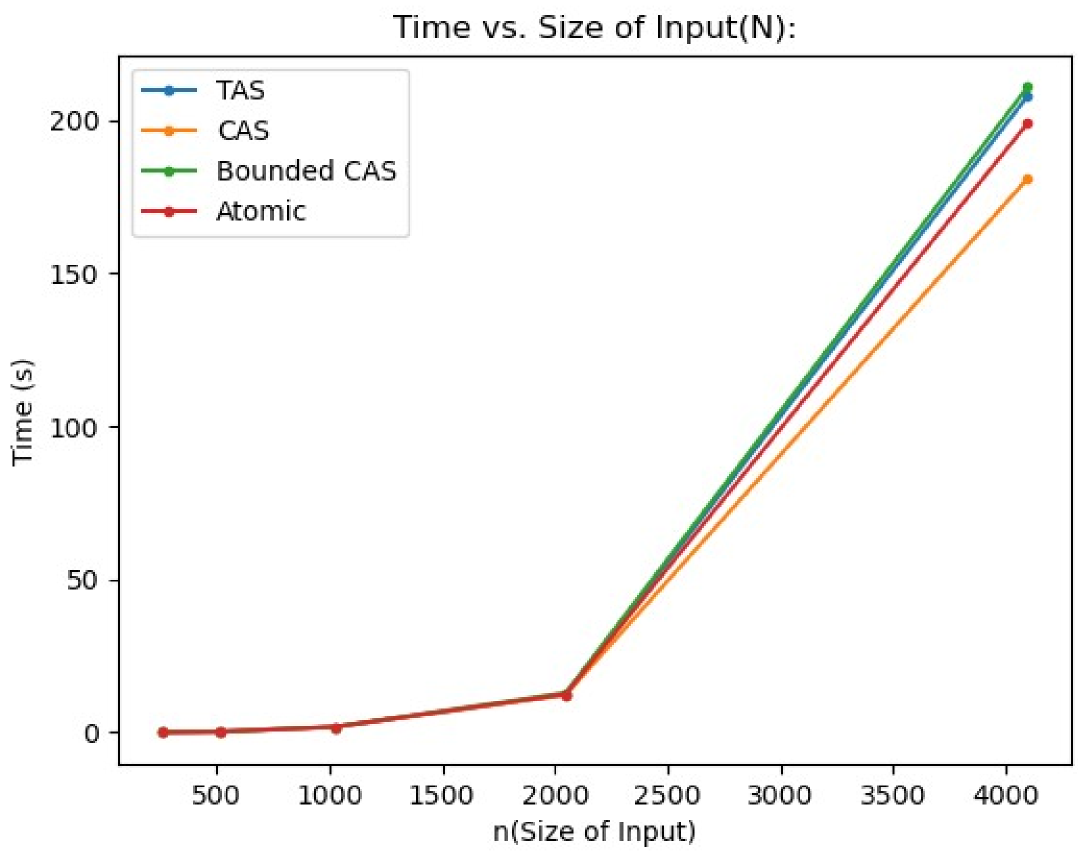
Recordings :

N	TAS	CAS	BCAS	ATINC
256	0.074	0.073	0.023	0.04
512	0.253	0.211	0.205	0.206
1024	1.785	1..792	1.839	1.818
2048	12.407	12.288	12.893	12.485
4096	207.946	180.963	210.871	199.02

Table 1: Execution Times for Different algos at various Input sizes

Graph:

figure 1 : Time vs Size of Input(N)



Observations: Time complexity appears to be a significant factor in the observed increase in execution time with larger input sizes across different algorithms.

Experiment 2:

Time (sec) vs rowInc:

Recordings :

rowInc	TAS	CAS	BCAS	ATINC
1	12.830	12.785	11.809	15.695
2	13.143	13.710	17.185	16.654
4	13.088	13.506	18.404	19.028
8	13.323	14.111	18.008	17.776
16	13.279	17.055	18.189	17.739
32	13.854	15.490	18.249	17.793

Table 2: Execution Times for Different rowInc Values

Graph :

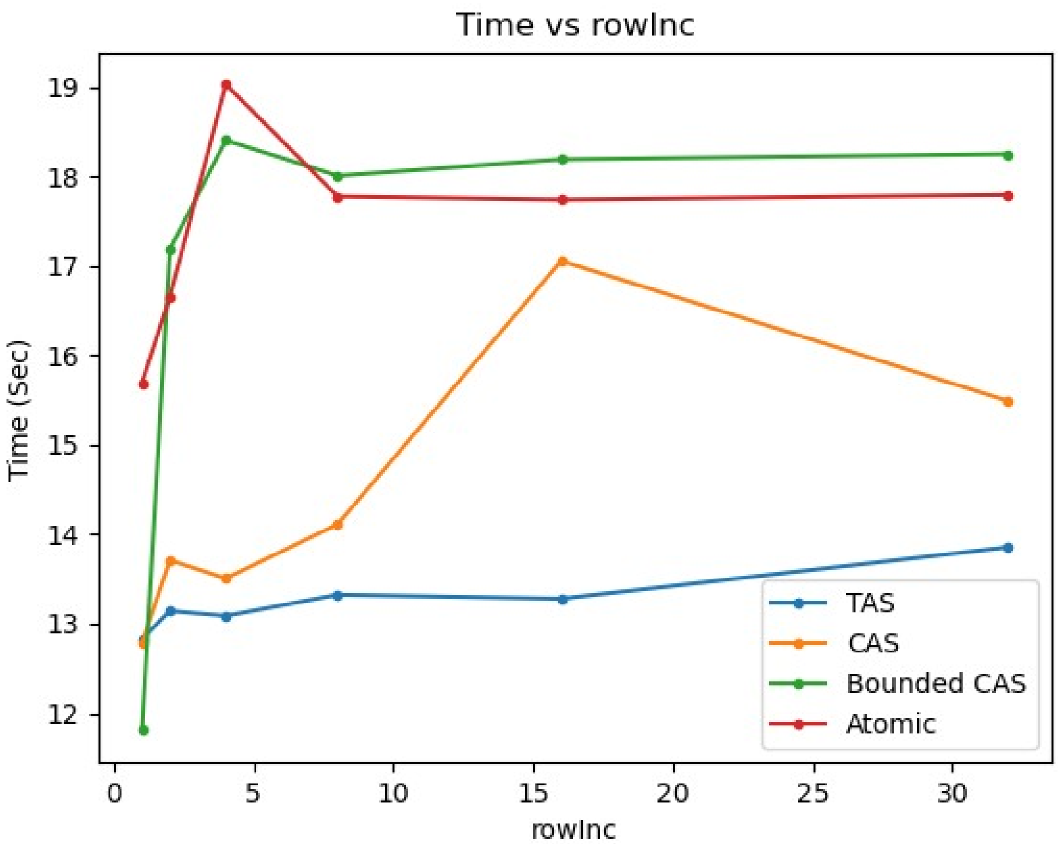


Figure 2 : Time vs rowInc

Observations : The relationship between execution time and changes in the row increment parameter appears to be erratic, showing no consistent pattern.

Experiment 3 :

Time (sec) vs Number of Threads (K) :

Recordings :

K	TAS	CAS	BCAS	ATINC
2	40.400	41.601	42.791	42.567
4	26.530	28.208	25.923	25.139
8	19.906	19.053	18.384	17.430
16	18.452	18.199	17.004	16.700
32	17.287	18.202	17.400	16.500

Table 3: Execution Times for Different (K) values

Graph :

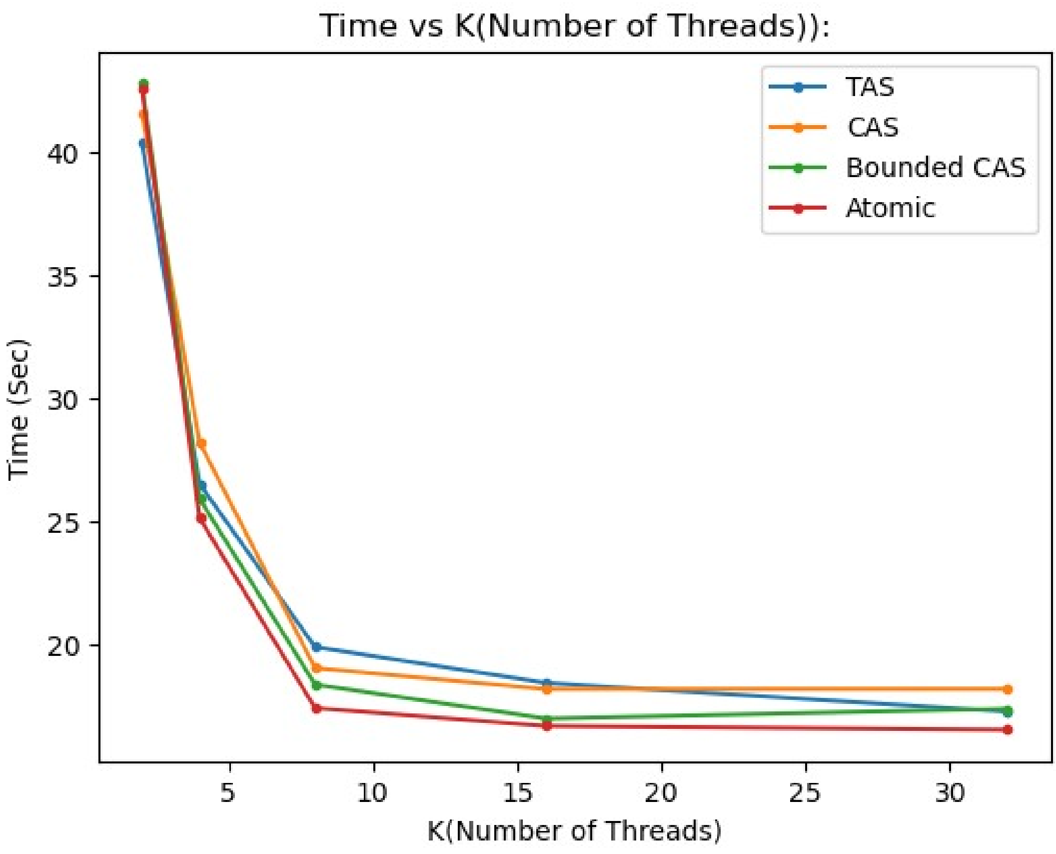


Figure 3 : Time (vs) Number of Threads (K)

Observations: The time taken decreases rapidly with an increase in the number of threads for all algorithms, reaching a saturation point where further increases do not significantly improve performance.

Experiment 4:

Time (sec) vs Algorithm:

Recordings :

Algorithm	Execution Time
Chunk	17.924
Mixed	17.937
TAS	18.615
CAS	17.429
BCAS	17.762
ATINC	18.117

Table 4 : Execution Times for Different Algos

Graph :

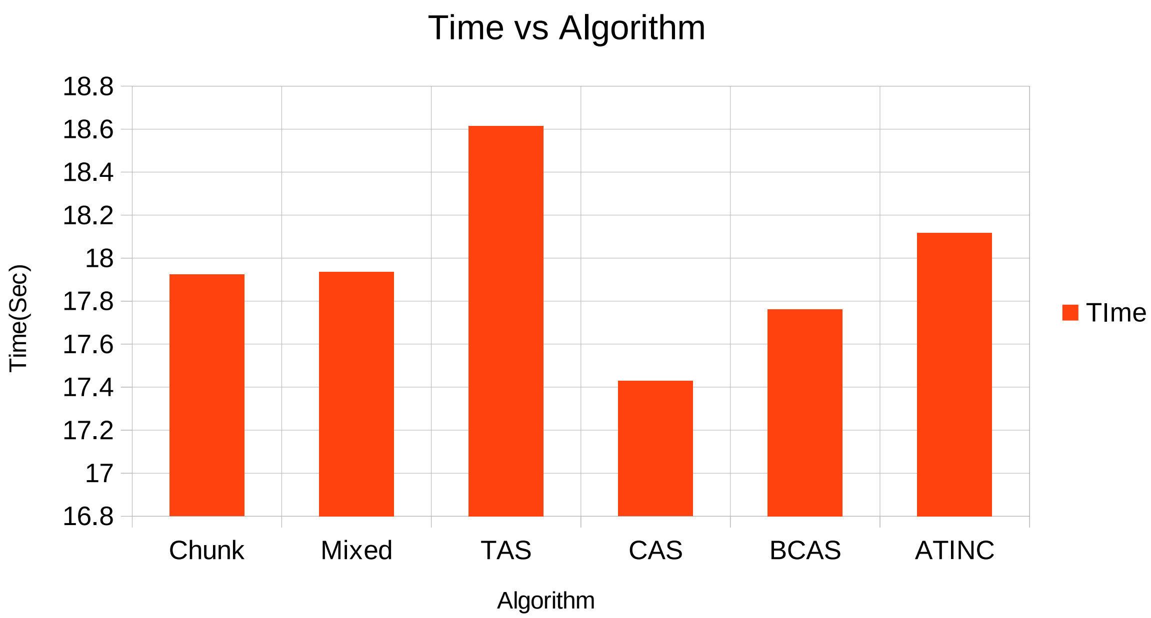


Figure 4 : Time vs Algorithm

Observations : All algorithms show similar overall performance, with dynamic algorithms slightly faster but BCAS and CAS slightly slower than TAS or Atomic increment.