ASSIGN3: DYNAMIC MATRIX SQUARING

Gaurav Choudekar - CS22BTECH11015

March 4, 2024

1 Introduction

This report is regarding programming assignment-3 conducted on dynamic matrix squaring through different mutual exclusion techniques.

2 About Code

The code is written in C++.Square matrix with number of threads and increment number is given as input with number of threads for calculation and resulting square matrix is obtained in output file.

3 Input format

A file is given as input,name of file is modified accordingly in code.N,K,rowincrement are given first in input files with spaces and then matrix numbers one by one,covering first all numbers from left to right from one row then going to lower one,row increment received is the chunk size for calculation by each thread.

4 Code Flow

4.1 Main Function

In the main function we are giving a file as input for which we want to obtain square matrix.we will then tokenise one by one elements of matrix from file and store them in global array. we will one by one call functions with threads for this different techniques individually.

4.2 Other Functions

We have different functions for different techniques for namely chunktas,chunkcas,chunkbcas,chunkatomic.multiplychunkta
are called by each thread created in each technique.

4.3 global variables used

- n:Represents the size of the square matrices involved in the matrix multiplication.
- k:Denotes the number of threads used for parallelizing matrix multiplication.
- result,resultm,resultbcas,resultatom:Matrices used to store the results of matrix multiplication using different techniques or strategies.
- given: Input matrices for matrix multiplication.
- step:Represents the step size used for dividing the computation into chunks.
- counter:Keeps track of the current index or position in the computation process.
- waiting:An array of booleans used for implementing some form of synchronization or coordination between threads.

- lockcas:An atomic integer used for locking in the critical section of the CAS (Compare-and-Swap) implementation.
- lockbcas:An atomic integer used for locking in the critical section of the bcas implementation.
- test:An atomic flag used in atomic operations.
- counteratom:An atomic integer likely used as a counter in atomic operations.

4.4 chunktas,chunkcas,chunkbcas,chunkatomic

These functions are directly called by main function.these functions further create pthreads and joins all of them to calculate time elapsed for particular technique.after joining all threads,data is written down into seperate files from respective result matrices.

- variables used
  - pthread threads[n]: Array to store pthread identifiers.
  - struct node* tempatomic[n]/tempbcas[n] etc: Array of node pointers to store thread-specific information.this is just created to test which thread is calculating which chunk of matrix
  - ofstream: Output file stream for storing the resulting matrix.
  - starttime,endtime,durationc:to calculate time elapsed
- Short introduction to techniques
  - TAS:TAS is a simple synchronization technique where a lock variable is set to a certain value (typically 1) when a thread wants to enter a critical section. Other threads querying this lock variable will keep looping until the lock becomes available
  - CAS:CAS compares the current value of a memory location with an expected value and, only if they are the same, modifies the memory location to a new value.CAS avoids the need for explicit locking and can provide better scalability in highly concurrent scenarios.
  - BCAS:Bounded CAS is a variant of CAS where there's a limit on the number of retries a thread can attempt to perform the CAS operation. This limit, or bound, helps prevent the thread from entering an infinite loop in case of continuous contention.
  - ATOMIC:Atomic operations provide low-level primitives for performing operations atomically without the need for explicit locking.They guarantee that an operation appears to occur instantaneously from the perspective of other threads, avoiding race conditions.

4.5 multiplychunktas

Each thread attempts to acquire a lock using Test-and-Set (TAS) mechanism before entering the critical section.testandset function assigns value true to lock and returns previous value.thread will be stucked in while loop till value of lock is 1 and pass through it if it was 0.only 1 thread will access its critical section in which it calculates its start and end row numbers for calculation.after critical section,it clears the lock,that is next any thread can enter its critical section.After this thread will complete its computations and store result in respective arrays and get ready to get assigned next chunk.This process stops after counter,that is number of rows assigned,equals or exceeds number of rows in input matrix.

4.6 multiplychunkcas

Threads attempt to acquire a lock using Compare-and-Swap (CAS) mechanism. Only one thread successfully updates the lock and enters the critical section.Within the critical section, each thread performs matrix multiplication on its assigned chunk of the matrix.After completing the computation, the thread releases the lock, allowing other threads to acquire it.compareexchangeweak return true only if value is changed,hence then only a thread will enter its critical section and calculate its starting and ending point and change lock value so that other thread can enter its critical section.After this thread will complete its computations and store result in respective arrays and get ready to get assigned next chunk.This process stops after counter,that is number of rows assigned,equals or exceeds number of rows in input matrix.

4.7 multiplychunkbcas

Inside this loop, the thread tries to acquire the lock using the BCAS mechanism. If successful, it proceeds to enter the critical section. If not, it keeps spinning until the lock is acquiredOnce inside the critical section, the thread performs its computation, which involves multiplying chunks of matrices. The computation is performed within the range specified by start and itssend.After completing its computation, the thread stores result in array and searches for another waiting thread to pass the lock to. This is done by iterating through the waiting array and finding the first waiting thread with an index greater than itsnum. If no such thread is found, it continues searching from index 0 to itsnum - 1.if still no thread is found,it itself goes for next chunk.this is insured by use of got which sees when should it for next loop waiting should be assigned true and when not.waiting for next thread is assigned false so that it enter critical section.

4.8 multiplychunkatomic

These function/technique use atomic operations (assuming counteratom is an atomic variable) to update the start index for the current thread. fetchadd() atomically increments counteratom by step and returns its previous value, which is assigned to start. Then, itssend is calculated as the end index for the current thread's portion of the computation. It calculates the product of the matrix elements specified by the current thread's start and itssend indices. The result is stored in the resultatom array.

5 Graph

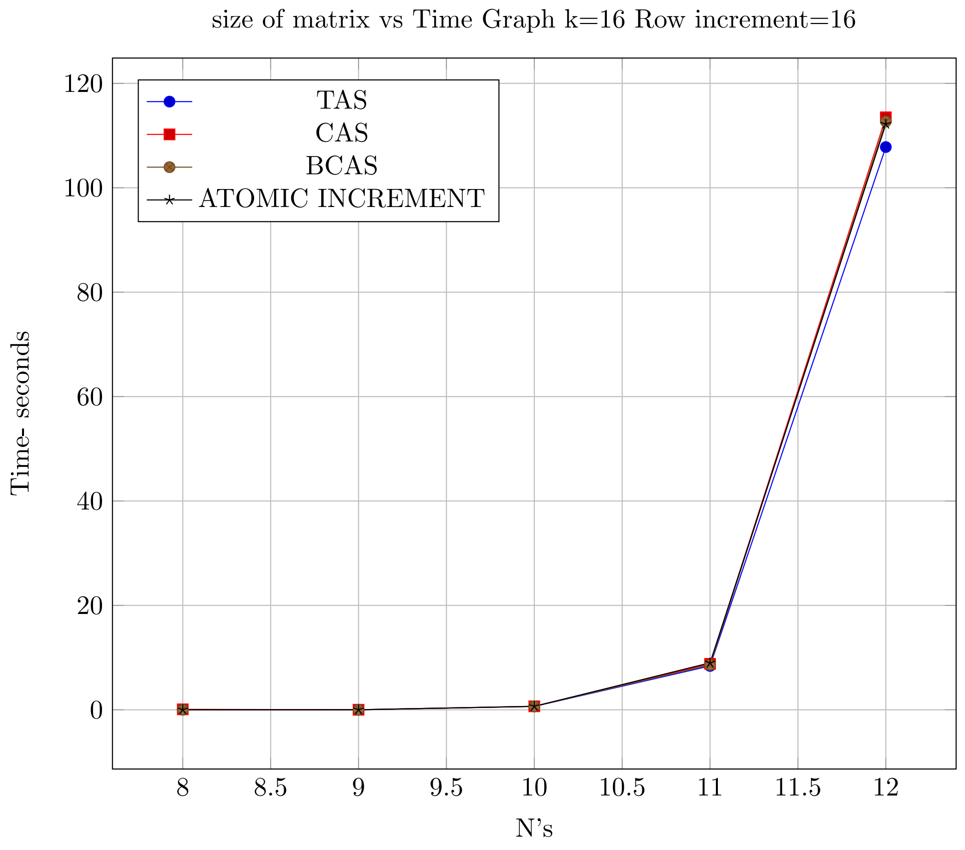

Figure 1: 2^N vs Time Graph

Table 1: Time Taken by Different Methods for Different Sizes of Matrices k=16 Row increment=16

Size of matrix-2^N	TAS	CAS	BCAS	ATOMIC INCREMENT
8	0.011	0.0092	0.0086	0.0084
9	0.0073	0.0074	0.0073	0.0071
10	0.65	0.67	0.67	0.66
11	8.4	8.8	8.6	9
12	107.8	113.5	112.7	112.2

6 Graph

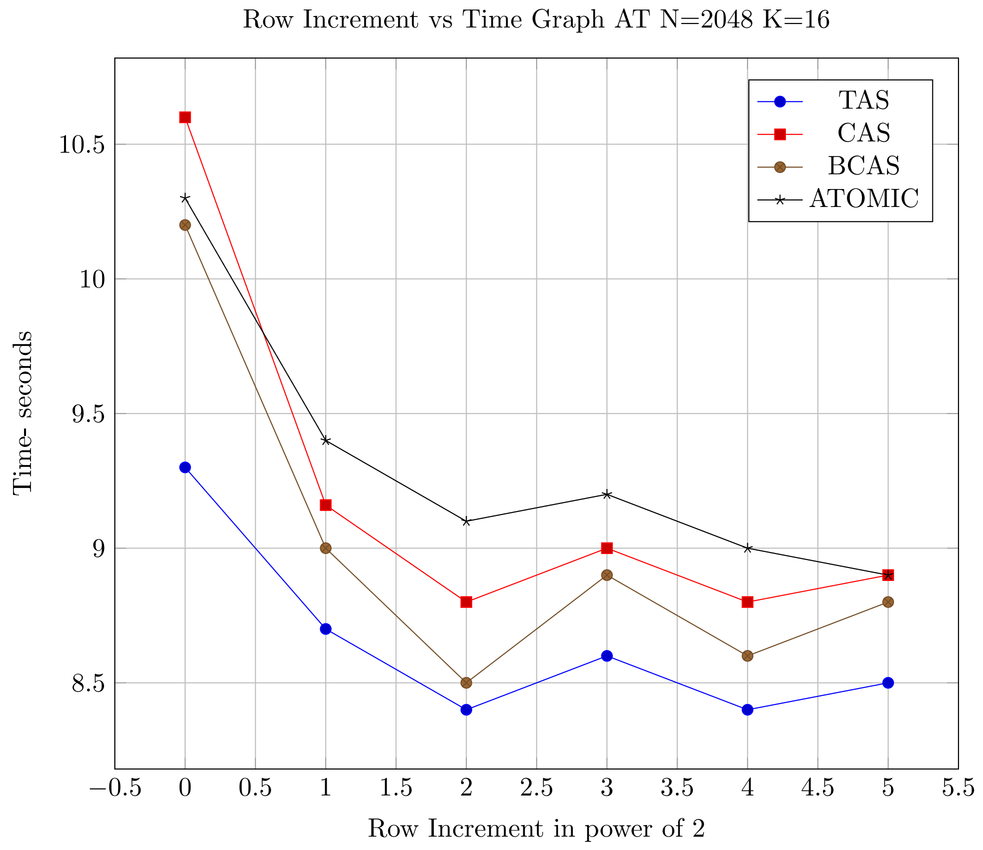

Figure 2: Row increment vs Time Graph

Table 2: Time Taken by Different Methods for Row Increment at $N = 2048$, $K = 16$

Row Increment	TAS	CAS	BCAS	ATOMIC
1	9.3	10.6	10.2	10.3
2	8.7	9.16	9.0	9.4
4	8.1	8.8	8.5	9.1
8	8.6	9.0	8.9	9.2
16	8.4	8.8	8.6	9.0
32	8.1	8.3	8.2	8.5

7 Graph

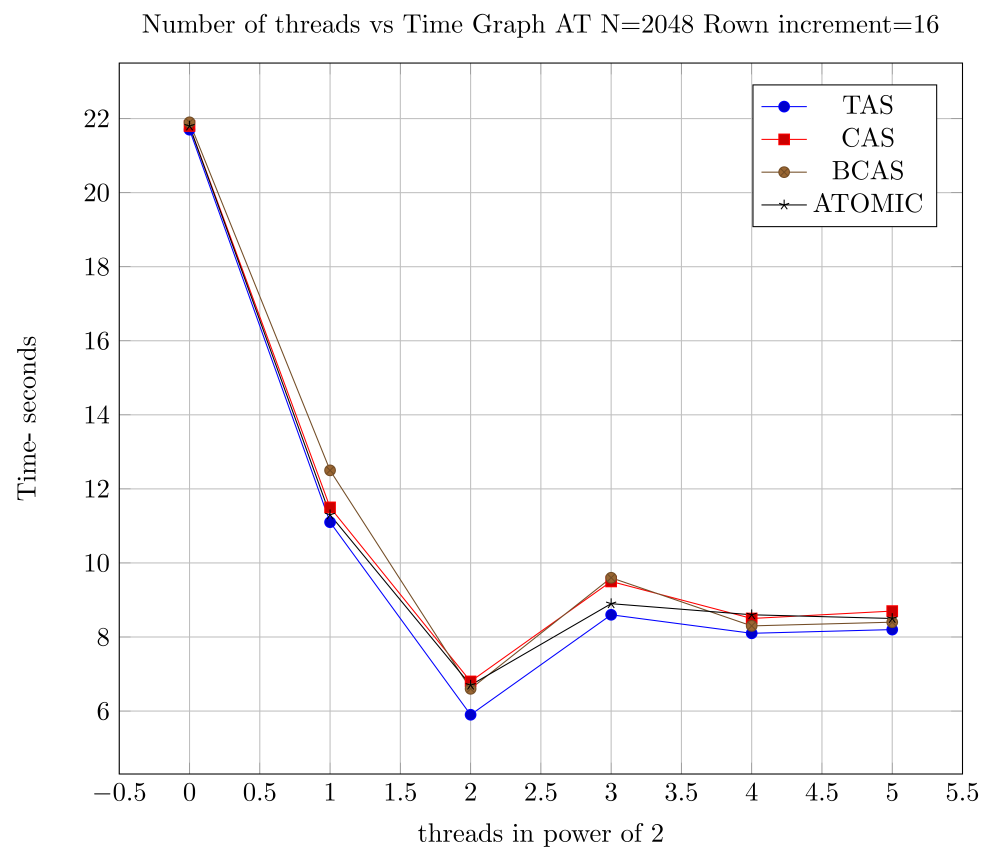

Figure 3: Number of threads vs Time Graph

Table 3: Time Taken by Different Methods for Threads at $N = 2048$, $rowincrement = 16$

Threads-2^K	TAS	CAS	BCAS	ATOMIC
0	21.7	21.8	21.9	21.8
1	11.1	11.5	12.5	11.3
2	5.9	6.8	6.6	6.7
3	8.6	9.5	9.6	8.9
4	8.1	8.5	8.3	8.6
5	8.2	8.7	8.4	8.5

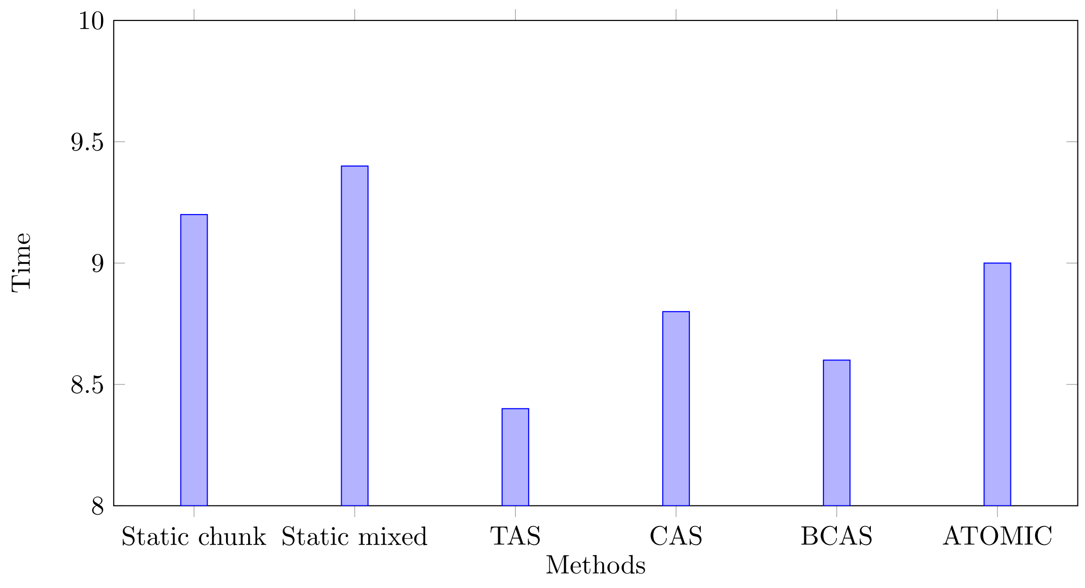

Figure 4: Time Taken by Different Algorithms at N=2048 K=16 Row increment=16

Methods	**Time**
Static chunk	9.2
Static mixed	9.4
TAS	8.4
CAS	8.8
BCAS	8.6
ATOMIC	9

Table 4: TIme taken by different algorithms at N=2048 k=16 row increment=16

8 output files

- output for TAS mutual exclusion method is saved with name ”chunka3tas.output”
- output for CAS mutual exclusion method is saved with name ”chunka3cas.output”
- output for BCAS mutual exclusion method is saved with name ”chunka3bcas.output”
- output for Atomic increment mutual exclusion method is saved with name ”chunka3atom.output”

9 Results and observations

9.1 Experiment 1:Time vs size

- All the curves have the same trend of increasing time with increasing size of matrix
- TAS seems to have the lowest execution time among all methods for most sizes, indicating its efficiency in handling concurrent access.
- CAS and BCAS shows almost same performance,BCAS is comparatively slighlty better.
- Atmoic starts with equal or little lower time for small N but grows rapidly for higher N.Its performance seems to be similar to or slightly better than BCAS, but slightly worse than TAS and CAS.

9.2 Experiment 2:Time vs Row increment

- Overall, all methods exhibit a decreasing trend in execution time as the row increment increases, suggesting that larger row increments generally lead to lower execution times.this decrease lowers for higher number of row increment.
- There is sudden and abnormal decrease in time at row increment=4 for all methods.
- TAS appears to be best on factor of row increment,having consistently lower execution times compared with other methods,while atomic generally appears to be worst,execution time difference atomic and CAS decreases with increasing number of row increment.
- BCAS appears to be better than CAS on factor of row increment.

9.3 Experiment 3:Time vs Number of threads

- TAS, CAS, BCAS, and ATOMIC all start with higher times for fewer threads and decrease as the number of threads increases.
- After certain number of threads,this doesnt decrease further and attains saturation after little increase.
- TAS exhibits the lowest time at each point, followed closely by BCAS and ATOMIC, with CAS having slightly higher times.
- FOr higher number of threads,BCAS appears to be better than CAS.
- for k=1 all methods have almost same time
- Performance of Atomic degrades compared to others for increasing number of threads.

9.4 Experiment 4:Time vs Algorithm

- Dynamic have better execution than static
- In dynamic,TAS has best execution time while atomic has worst from my experiment
- In static,row increment has better than mixed
- BCAS has better execution time than CAS

9.5 General observations over all experiments

- TAS has least execution time in dynamic methods.
- BCAS is better than CAS.
- performance of ATOMIC degrades with higher number of threads and row increment.
- improvement due to increase in number of threads and row increments halts after certain limit and saturates or may be degrades.

10 Analysis and Conclusions

- As size of matrix increases,more computation is needed to be done,hence time increases with increasing N.
- With a higher row increment, fewer synchronization operations are needed, leading to reduced overhead.while fine-grained synchronization(less row increment) can lead to better scalability, particularly in scenarios with high contention, as threads spend less time waiting for access to shared resources.this is tradeoff between high and low row increment hence performance saturates after certain range and doesnt become better throughot number line.

- For less threads cpu utilisation is less but for higher number if threads,context switching increases time,hence performance saturates after certain range for all methods.
- CAS relies on a simple busy-waiting mechanism to contend for locks, BCAS employs a more intelligent approach by backing off when contention is detected.hence improved perfomance is observed in BCAS compared to CAS.
- As matrix size increases, the likelihood of contention among threads accessing shared data structures also increases. Atomic operations may introduce contention as threads compete for access to atomic variables, leading to serialization and increased waiting times, especially for larger matrices.
- For smaller matrices, where the computation time is relatively short and the number of concurrent accesses to shared data is limited, atomic operations can provide synchronization without incurring significant overhead.
- Since TAS involves a single atomic operation, it tends to have lower overhead compared to techniques that require additional checks or retries,hence looks better than other methods at many places
- Dynamic allocation allows you to distribute rows of the matrix dynamically among threads based on the current workload. This enables better load balancing, as threads can handle variable-sized chunks of data depending on the availability of computational resources.
- When threads are assigned fixed-sized chunks of data, contention may arise if some chunks require more computational effort than others. Dynamic allocation enables threads to request rows on-demand, reducing contention for specific chunks of data and potentially mitigating bottlenecks caused by uneven workload distribution.
- As for our experiment,we conclude that dynamic matrix squaring is better than static.
- Different mutual exclusion techniques have its different pros and cons
- Different mutual exclusion techniques may have little different behavior on different conditions such as increasing size of matrix or increasing number of threads etc.
- Application of mutual exclusion technique must be considered to have maximum benefit of dynamic technique
- environment on which the application runs also plays important role.