Aayush Kumar
CO21BTECH11002
Assignment 3

AtomicMRMWRegister class:

This class is designed to provide atomic read and write operations for a shared register that can be accessed by multiple threads concurrently. The register is implemented as a vector of pairs, where each pair contains an integer value and a timestamp.
The class AtomicMRMWRegister contains a private member variable ‘arr’, which is a vector of pairs of integers. Each pair represents a value and a timestamp. The constructor AtomicMRMWRegister(int n) takes an integer n as an argument and initializes the vector arr with n elements, each initialized with {0, 0}. This effectively creates an array of size n, where each element is initially set to value 0 with a timestamp of 0.
The write method is used to write a new value v into the register at a specified index id. It first initializes a pair p to {0, 0}. This pair will be used to keep track of the most recent value in the register. The method then iterates through the entire arr vector to find the pair with the highest timestamp (most recent value). It updates the p pair accordingly. Finally, it updates the element at index id in arr with the new value v and a timestamp that is one greater than the timestamp of the most recent value.
The read method is used to read the most recent value from the register. Similar to the write method, it initializes a pair p to {0, 0} and iterates through the arr vector to find the pair with the highest timestamp (most recent value). It returns the value of the most recent pair, which represents the most recent value written to the register.

testAtomic function:

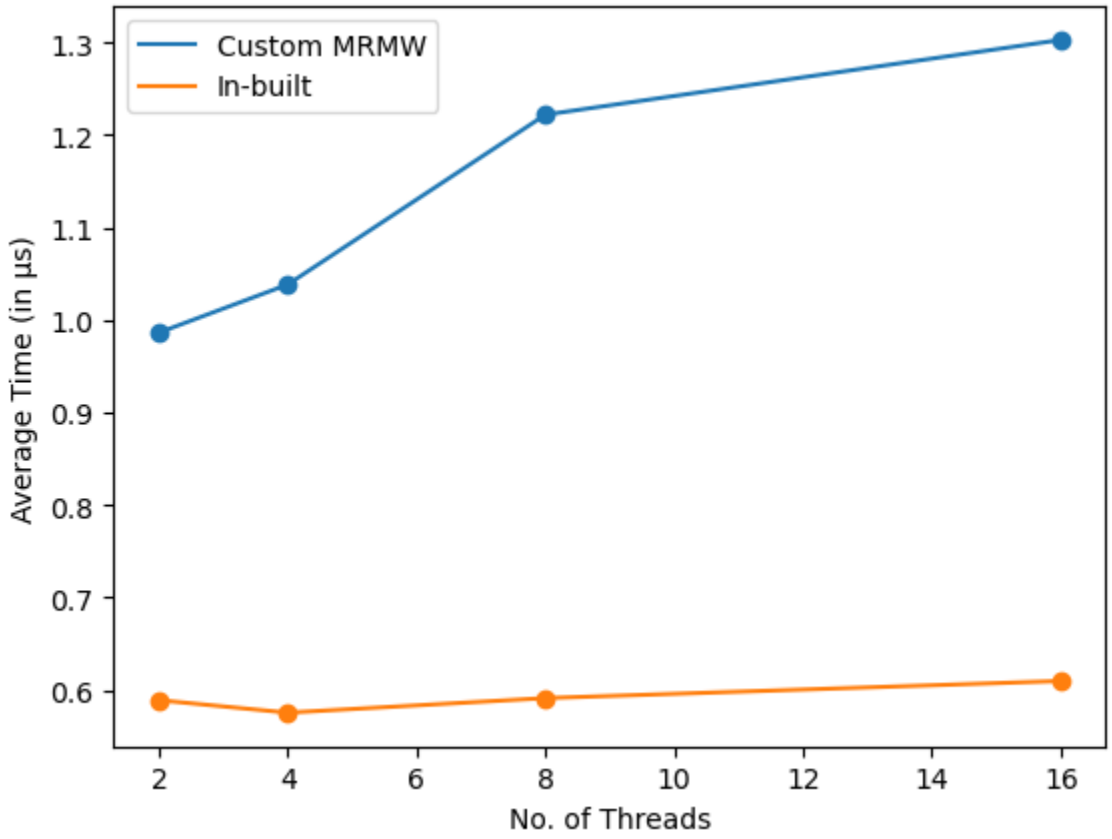
This function is called by each thread. It simulates read and write operations. It also calculates the time taken by each thread to complete numOps operations and stores the read and write time taken by each thread in readTime and writeTime respectively. To ensure that the threads are not synchronized, each thread sleeps for a random time after each operation. The random time is generated using exponential distribution with mean lambda. The random number generator is seeded with the thread id and time(NULL). Hence, each thread has a different random number generator. The random number generator is used to generate a random number between 0 and 1. If the random number is less than 0.5, then read operation is performed, else write operation is performed.

main() function:

The main function reads the input from the file and creates n threads. It then waits for all the threads to finish and then computes the average time taken by each thread to complete numOps operations for both custom MRMW register and in-built register.

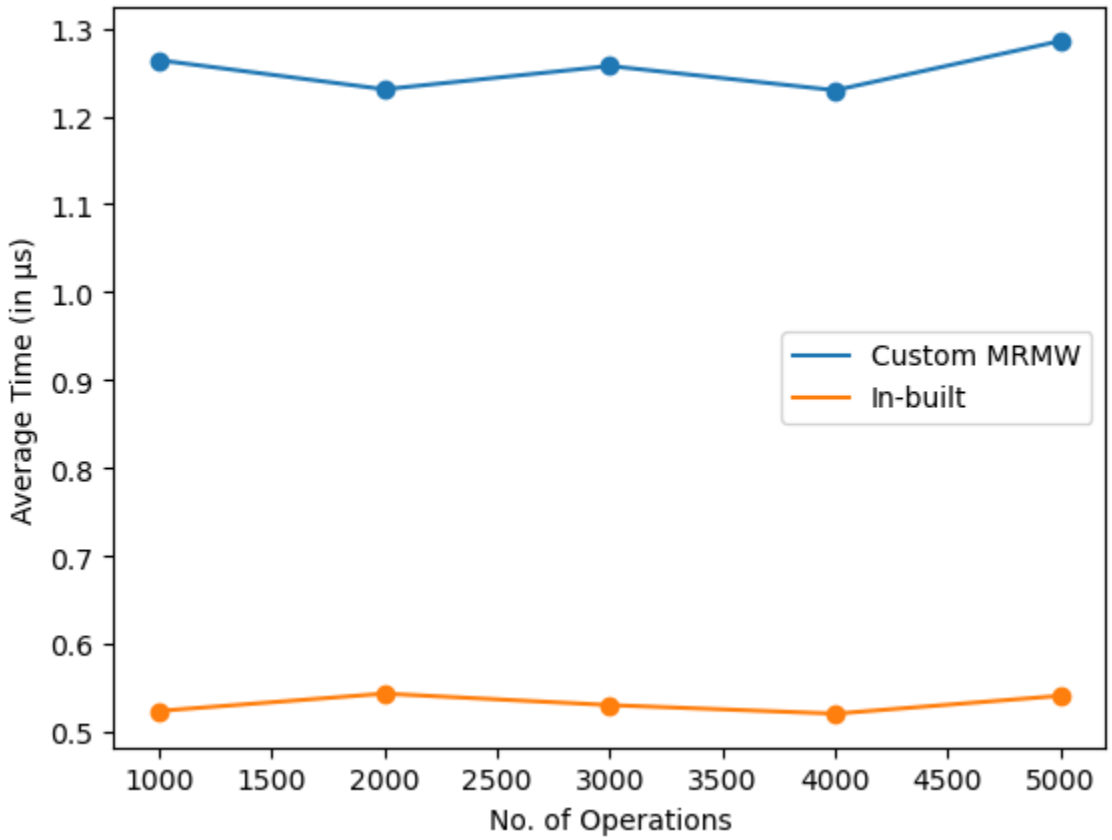
Results:

Average time vs number of threads (keeping numOps = 5000, lambda = 5)



- As the number of threads increases, the time taken for custom MRMW increases as well since the read and write operations are proportional to the number of threads. Time for in-built MRMW also increases but the increase is much lower in comparison to the custom MRMW.
- Custom atomic MRMW is slower than the in-built atomic MRMW for any number of threads.

Average time vs number of operations (keeping capacity = 16, lambda = 5)



- On increasing the number of operations, the average time remains almost the same for both custom MRMW and in-built MRMW. This is because read and write operations depend only on the number of threads and are wait free.
- Custom atomic MRMW is slower than the in-built atomic MRMW for any number of operations.