Implementing Atomic MRMW Register from M-Valued Regular Registers

Design

Custom Atomic variable:

- AtomicMRMW class was implemented in CPP as given in the textbook("The Art of multiprocessor programming")
- To satisfy the underneath properties of the variables of the class, additional classes were implemented (AtomicMRSWRegister, AtomicSRSWRegister)
- AtomicMRMW assumes AtomicMRSWRegisters to be available to work with, AtomicMRSW assumes AtomicSRSWRegisters are available and AtomicSRSW assumes RegularSRSW to be available
- We assume that CPP variables are RegularSRSW
- All of the above classes were implemented by taking reference from the textbook

CPP Atomic variable:

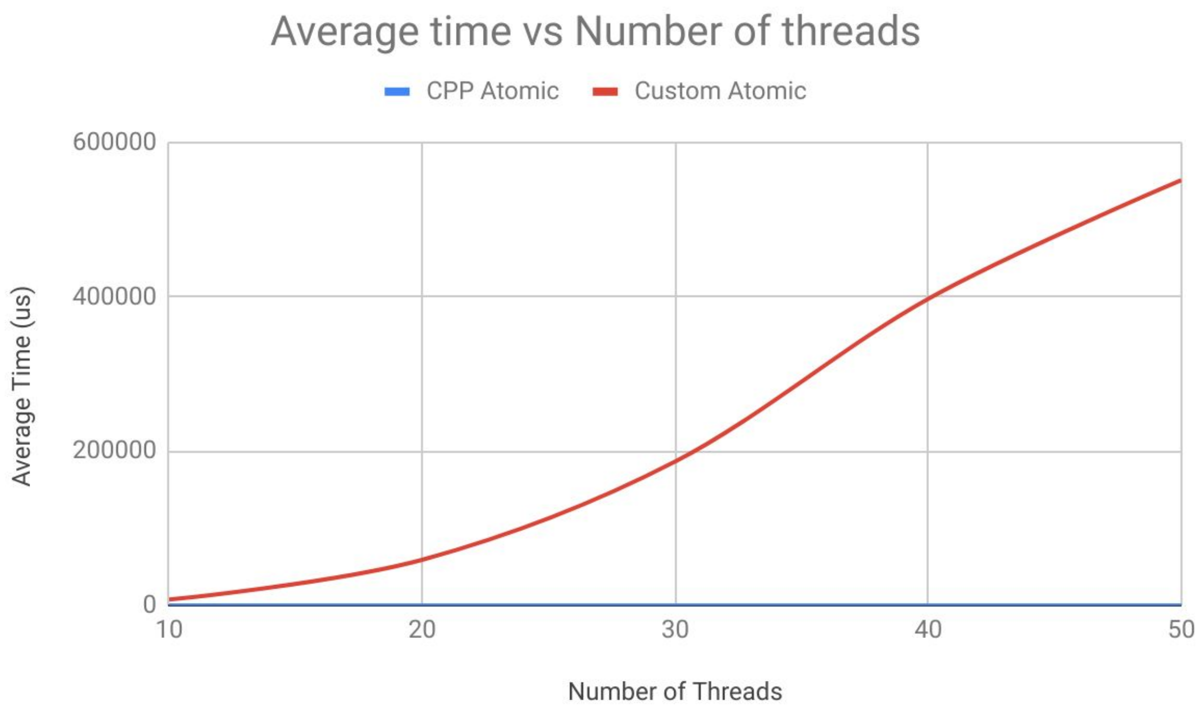
- atomic<int> variable was used from the atomic library included in the atomic.h header
- atomic_load and atomic_store was use to read and write to the variable respectively

Graphs and Analysis

lamda = 5
K = 100 (Number of time to execute actions)
N = 10 to 50 (steps of 10)

★ Each test was run 5 times and the average value was calculated to plot the graphs

Threads	CPP Atomic(us)	Custom Atomic(us)
10	3.8	7457.6
20	6.2	59024.2
30	3.4	186335.2
40	7.4	397139.6
50	11.2	550804.6



- As can be observed from the above graph, the time taken to run the custom implementation of the atomic variable takes a lot of time and is largely dependant on the number of threads, having a time complexity of O(Number of threads)
- This can be explained by the fact that memory is being allocated dynamically to create object for reading and writing in AtomicMRMW class and for writing in AtomicSRSW class
- With each read and write action 3 malloc(new in cpp) actions are performed, this is done to satisfy the underneath properties of the variables assumed in the implementation of a class as discussed before
- The CPP atomic variable on the other hand does seem to have very little to no dependency on the number of threads and perform extremely well compared to the custom implementation, it is possible that a lot of optimizations were done in the IR code level to improve the efficiency and provide the atomic guaranteed to the variable

Correctness of Custom Atomic implementation

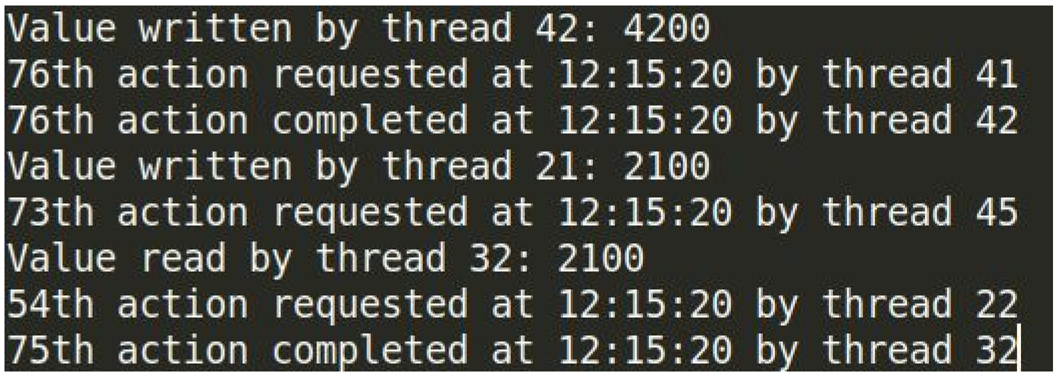
We can confirm the correctness of the program by observing the three properties given in the textbook for atomic variables

- No read returns a value from the future

$$R^i \rightarrow W^i \text{ (Not possible)}$$

- No read call returns a value from the distant past

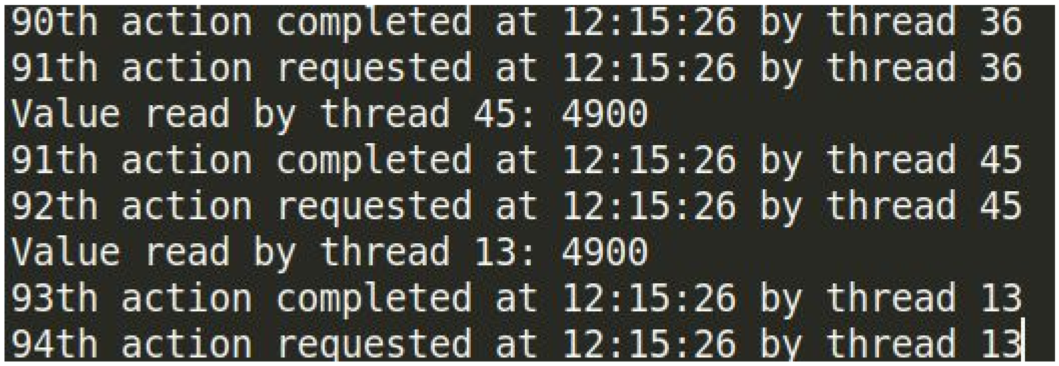
$$W^i \rightarrow W^j \rightarrow Ri \text{ (Not possible)}$$



```
Value written by thread 42: 4200
76th action requested at 12:15:20 by thread 41
76th action completed at 12:15:20 by thread 42
Value written by thread 21: 2100
73th action requested at 12:15:20 by thread 45
Value read by thread 32: 2100
54th action requested at 12:15:20 by thread 22
75th action completed at 12:15:20 by thread 32
```

- An earlier read cannot return a value later than that returned by a later read

$$R^i \rightarrow R^j \ \ (i <= j)$$



```
90th action completed at 12:15:26 by thread 36
91th action requested at 12:15:26 by thread 36
Value read by thread 45: 4900
91th action completed at 12:15:26 by thread 45
92th action requested at 12:15:26 by thread 45
Value read by thread 13: 4900
93th action completed at 12:15:26 by thread 13
94th action requested at 12:15:26 by thread 13
```

The above snippets were taken from the log file of the custom atomic implementation