Implementing Multi Reader Multi Writer Register
CO21BTECH11004

Implementation: -

- Global variables lambda, capacity, and numOps are created.
- Global array threadTime is dynamically allocated in the main of size number of threads and contains time for read/write operations per thread.
- C++ threads are used, creating ‘capacity’ number of threads, which calls the testAtomic function.
- Time is measured in microseconds using the chrono library.
- Random number to decide the read/write operation is generated using uniform_real_distribution, while that for sleep is generated using exponential_districution using random library.

MRMW atomic register: -

- stampedValue class is implemented, given in Fig 4.10.
- atomicMRMWRegister class is implemented, given in Fig 4.14.
- Global object of class atomicMRMWRegister<int> shVar is declared.
- In the testAtomic function, to perform read operation, call shVar.read(), and for write operation, call shVar.write(value, threadID).

Inbuilt Atomic: -

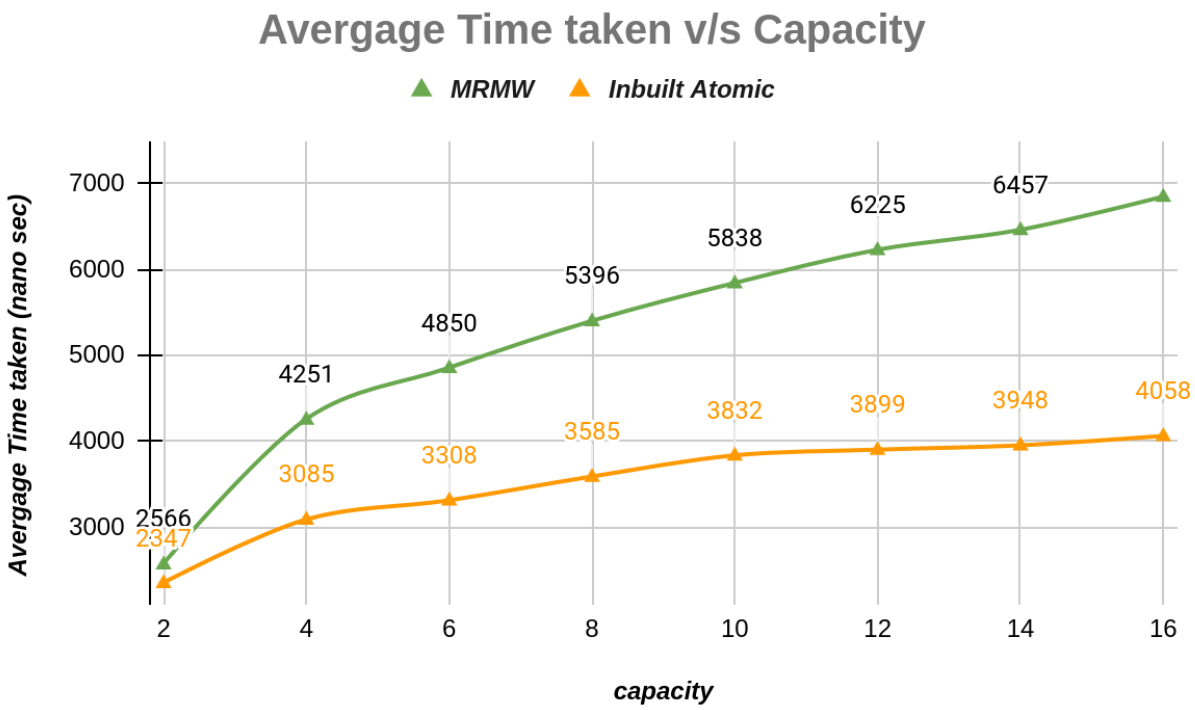
- A global variable, atomic<int> shrVar, is declared.
- In the testAtomic function, to perform read operation, call shrVar.load(); for write operation, call shrVar.store(value).

LogFile: -

- Four print statements in the testAtomic function are printed by each thread numOps times. Time in the log file is in nanoseconds with respect to start time declared globally.
- MRMW atomic register: - ‘logFileMRMW.txt’
- inbuiltMRMW: - ‘logFileInbuiltMRMW.txt’

Plot

Impact of average time with increasing Capacity



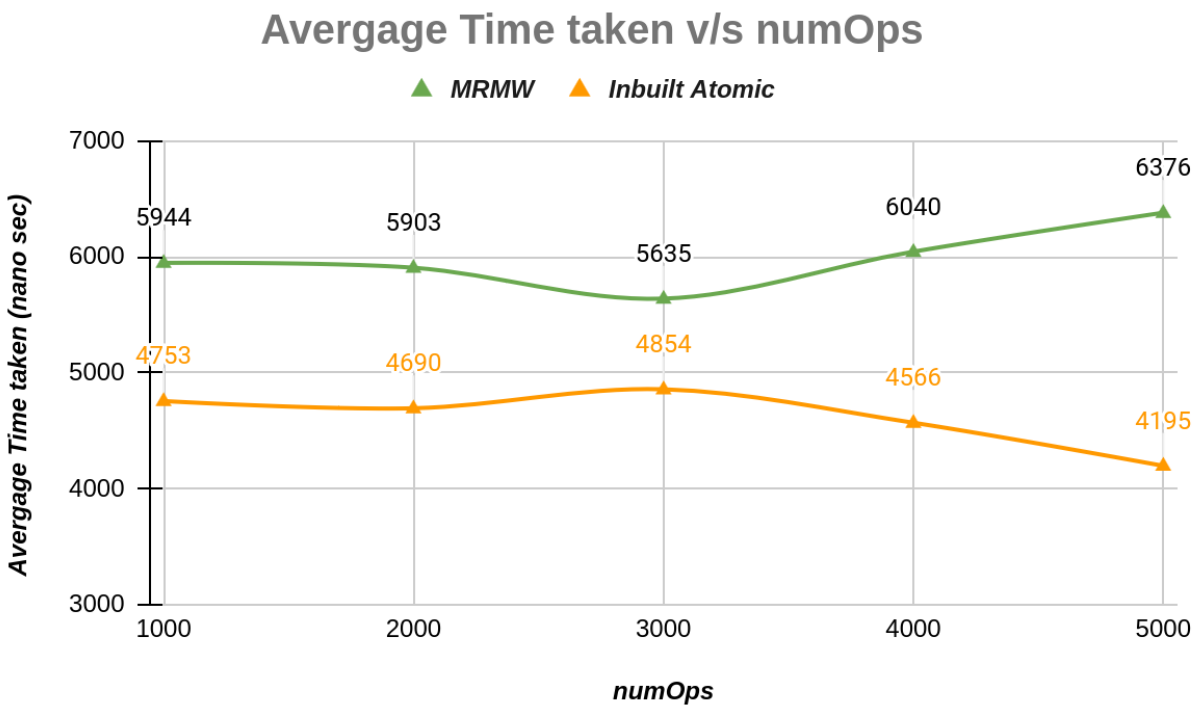
- For MRMW, the average time taken per operation increases with an increase in the number of threads because the algorithm has time complexity O(n), where n is the number of threads, so as thread increases, more time taken per operation.
- For inbuilt atomic, the average time taken per operation increases initially, and then the rate of increase decreases.
- Inbuilt atomic have less average time taken per operation as compared to MRSW implementation.

Data used for plotting graph

(MRMW) capacity	Average Time_taken (nanoseconds)					Time average
	1st	2nd	3rd	4th	5th	
2	2579	2473	2457	2426	2894	2566
4	4324	4369	3998	4087	4477	4251
6	4024	4557	6014	4353	5304	4850
8	5385	6044	5185	5087	5279	5396
10	5766	5887	5929	5557	6049	5838
12	6484	5992	6737	5862	6047	6225
14	6237	6055	6721	6785	6486	6457
16	6580	7185	6445	7128	6879	6843

(Inbuilt) capacity	Average Time_taken (nanoseconds)					Time average
	1st	2nd	3rd	4th	5th	
2	2424	2316	2494	2114	2384	2347
4	2993	3550	2667	2629	3587	3085
6	3359	3279	3758	2770	3374	3308
8	3548	3830	4135	3387	3025	3585
10	4045	3842	4215	3296	3760	3832
12	4226	3915	3837	3452	4063	3899
14	3802	3533	3949	4130	4328	3948
16	4113	4094	4063	3794	4226	4058

Plot
Impact of average time with increasing numOps



- For MRMW, the average time taken per operation almost remains constant but slightly increases at the end. This is because as the number of operations increases, total time also increases. Hence time per operation would remain constant. (5.6 micro sec to 6.3 micro sec)
- For inbuilt atomic also, time taken per operation almost remains constant, here slight decrease at end. Same here also, with the increase in the number of operations, time also increases, so time per operation remains constant. (4.8 micro sec to 4.2 micro sec).
- Inuilt atomic performs better than MRMW with respect to time as MRMW has linear time complexity and is not very efficient.

Data used for plotting graph

(MRMW) numOps	Average Time_taken (nanoseconds)					Time average
	1st	2nd	3rd	4th	5th	
1000	5386	5424	6370	6335	6208	5944
2000	5100	5735	6959	6737	4983	5903
3000	6339	5690	5395	5272	5481	5635
4000	5615	7258	4723	5040	7564	6040
5000	6181	6469	6213	6391	6627	6376

(Inbuilt) numOps	Average Time_taken (nanoseconds)					Time average
	1st	2nd	3rd	4th	5th	
1000	4850	4035	4226	5226	5429	4753
2000	3955	4639	4665	5132	5060	4690
3000	4503	4515	5106	5455	4692	4854
4000	4908	4454	4644	4372	4454	4566
5000	3876	4518	4103	4150	4327	4195