MULTI-CORE PROGRAMMING

PROJECT – 3

IMPLEMENTATION OF CONCURRENT QUEUES

Team Members:

Rahul Aravind Mehalingam(rxm151730)

Praveen Erode Murugesan(pem150030)

Problem Statement:

The aim of the project is to implement the Concurrent Queues using different algorithms such as

I. Lock-Based Algorithm
II. Lock-Free Algorithm

Experimental Setup:

CPU cores: 16

Operating System: CentOS

Programming Language: Java

No of threads experimented: 2 to 64

Correctness Test Strategy:

Verifying the correctness of the concurrent data structure proved to be a NP-Hard problem.

Threads are given thread ID. Three threads namely t1, t2, t3 performs enque operation concurrently. Each enque operation adds a tuple of Thread ID and an integer number [Each thread enques number from 1 to n]. Once the enque operations are completed by these threads, only one thread performs deque operation. A single dimensional vector is created where the indexes attribute to the thread Id and the value attribute to the count for each thread. The count values are initialized to zero. Whenever the thread deques an item (a tuple), the count value corresponding to the particular thread is updated after checking whether the new key is one value greater than the previous key. If the difference between the previous value of the thread stored in the vector and the current value after the main thread deque is not equal to one, then the concurrency correctness is incorrect with the implementation.

Note - Sample output included at the last

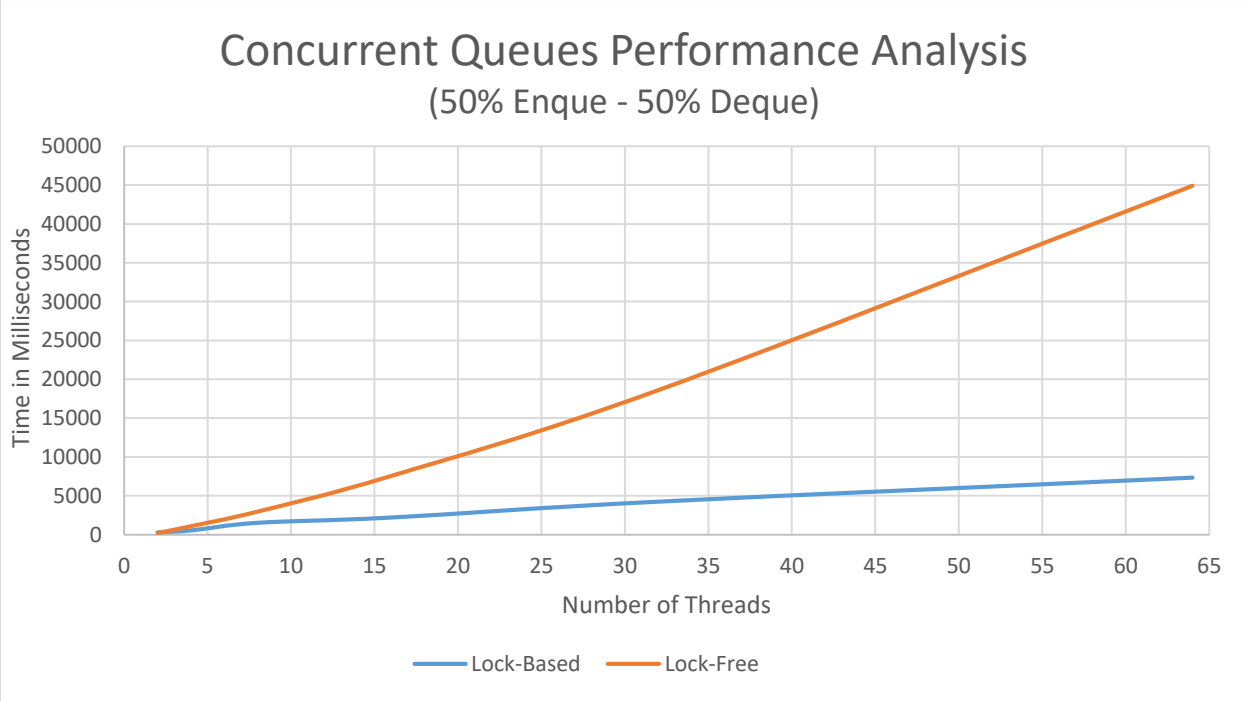
Testing in TACC Machine (Performance Training):

The implemented concurrent queues are tested with high load and their results are tabulated and pictographically represented as follows. Each list is subject to 1Million operations by keeping the key space bounded to 1000. The performance of the lists is analyzed by varying the number of threads.

50% Enque – 50% Deque:

This flavor of test setup had 50% enque operations, 50% deque operations. The performance is as follows:

Number of threads	Performance of Algorithms	
	Lock-Based	**Lock-Free**
2	311	224
4	564	1096
8	1542	2986
16	2218	7575
32	4259	18615
64	7349	44926



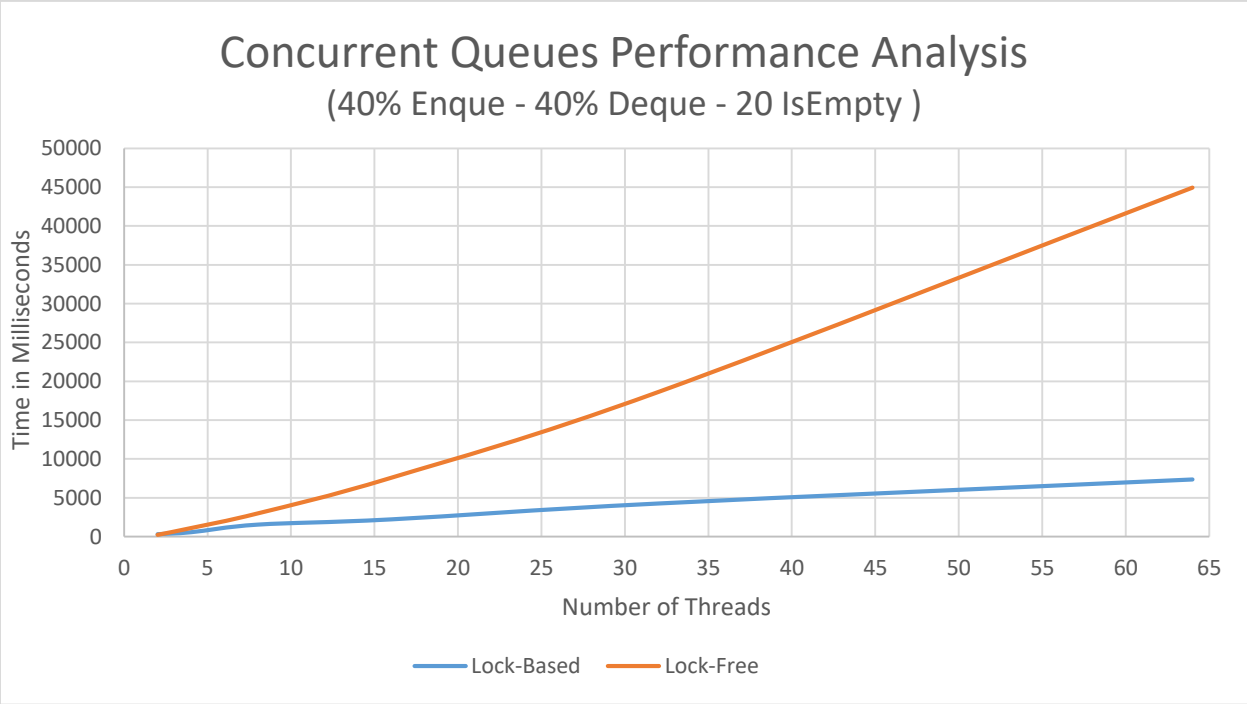
Inference:

It is seen that lock-based queue algorithm performs better if the data structure needs to be used in 50% Enque – 50% Deque scenario.

40% Enque – 40% Deque – 20% IsEmpty:

This flavor of test setup had 40% enque operations, 40% deque operations and 20% IsEmpty operations. The performance is as follows:

Number of threads	Performance of Algorithms	
	Lock-Based	**Lock-Free**
2	260	481
4	489	2816
8	1113	4360
16	1981	6767
32	3417	15447
64	19226	33020



Inference:

It is seen that lock-based algorithm beats the lock-free algorithm in performance by a huge margin for this kind of work load on concurrent queues. The lock-free algorithm is outperformed by the lock based algorithm probably because of the outer most while loop that gets executed many number of time because of the failure of CAS. In case of enque, failure of CAS of last.next and in case of deque, failure of CAS of head causes the while loop to repeat number of times until corresponding CAS succeeds whereas in the lock based algorithm, same steps are not repeated these many times to perform a single operation. Example: In case if 5 threads tries to enque, only one thread succeeds whereas the others fail. So the next time all the other 4 threads try executing all steps in the while loop and even then one thread succeeds and the other three fails. So this gets repeated until all threads complete their respective operation. This

results in taking O(n^2) time (since every time only one thread succeeds and rest iterated over the while loop) rather than O(n) time. So in these cases, lock-based algorithm works way better than the lock-free algorithms.

Conclusion:

The concurrent queues using different algorithms are implemented and their performances are analyzed and compared to infer the best suited implementation for different workloads.

Output for Correctness Testing:

****

********Test Statistics********************

Number of Threads: 4

Concurrent Queue Lock based Synchronization

Thread Id 3 Enqueue 1

Thread Id 2 Enqueue 1

Thread Id 1 Enqueue 1

Thread Id 1 Enqueue 2

Thread Id 3 Enqueue 2

Thread Id 3 Enqueue 3

Thread Id 1 Enqueue 3

Thread Id 2 Enqueue 2

Thread Id 2 Enqueue 3

Thread Id 4 Enqueue 1

Thread Id 4 Enqueue 2

Thread Id 4 Enqueue 3

Time: 2 msec.

Memory: 4 MB / 128 MB.

4 threads have finished their enque operations. No of Operations 3

Main Thread dequed key 1 which was enqued by Thread Id 3

Main Thread dequed key 1 which was enqued by Thread Id 1

Main Thread dequed key 1 which was enqued by Thread Id 2

Main Thread dequed key 2 which was enqued by Thread Id 1

Main Thread dequed key 2 which was enqued by Thread Id 3

Main Thread dequed key 3 which was enqued by Thread Id 3

Main Thread dequed key 3 which was enqued by Thread Id 1

Main Thread dequed key 2 which was enqued by Thread Id 2

Main Thread dequed key 3 which was enqued by Thread Id 2

Main Thread dequed key 1 which was enqued by Thread Id 4

Main Thread dequed key 2 which was enqued by Thread Id 4

Main Thread dequed key 3 which was enqued by Thread Id 4

Concurrency correctness PASSED!!!

****

Concurrent Queue Lock free Synchronization

Thread Id 1 Enqueue 1

Thread Id 1 Enqueue 2

Thread Id 2 Enqueue 1

Thread Id 2 Enqueue 2

Thread Id 2 Enqueue 3

Thread Id 1 Enqueue 3

Thread Id 4 Enqueue 1

Thread Id 4 Enqueue 2

Thread Id 4 Enqueue 3

Thread Id 3 Enqueue 1

Thread Id 3 Enqueue 2

Thread Id 3 Enqueue 3

Time: 1 msec.

Memory: 6 MB / 128 MB.

4 threads have finished their enque operations. No of Operations 3

Main Thread dequed key 1 which was enqued by Thread Id 1

Main Thread dequed key 1 which was enqued by Thread Id 2

Main Thread dequed key 2 which was enqued by Thread Id 1

Main Thread dequed key 2 which was enqued by Thread Id 2

Main Thread dequed key 3 which was enqued by Thread Id 2

Main Thread deqed key 3 which was enqued by Thread Id 1

Main Thread deqed key 1 which was enqued by Thread Id 4

Main Thread deqed key 2 which was enqued by Thread Id 4

Main Thread deqed key 3 which was enqued by Thread Id 4

Main Thread deqed key 1 which was enqued by Thread Id 3

Main Thread deqed key 2 which was enqued by Thread Id 3

Main Thread deqed key 3 which was enqued by Thread Id 3

Concurrency correctness PASSED!!!

****

Sample Output of Performance Training:

```
*******Test Results************
50% Enque 50% Deque
Concurrent Queue Lock based Synchronization
Number of Threads: 2
Number of Operations: 3
Time: 2 msec.
Memory: 2 MB / 128 MB.
*****************************************
Concurrent Queue Lock based Synchronization
Number of Threads: 4
Number of Operations: 3
Time: 0 msec.
Memory: 2 MB / 128 MB.
*****************************************
Concurrent Queue Lock based Synchronization
Number of Threads: 8
Number of Operations: 3
Time: 1 msec.
Memory: 2 MB / 128 MB.
*****************************************
*****************************************
Concurrent Queue Lock free Synchronization
Number of Threads: 2
Number of Operations: 3
Time: 1 msec.
Memory: 3 MB / 128 MB.
*****************************************
Concurrent Queue Lock free Synchronization
Number of Threads: 4
Number of Operations: 3
Time: 1 msec.
Memory: 3 MB / 128 MB.
*****************************************
Concurrent Queue Lock free Synchronization
```

```
Number of Threads: 8
Number of Operations: 3
Time: 2 msec.
Memory: 3 MB / 128 MB.
*****************************************
*****************************************
```