**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

1. Lock-Based Algorithm
2. 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. 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. This deque operation maintains a vector of 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 value is one greater than the previous one. If there is a mis-match then it is found that there is something wrong with the implementation.

*Note - Sample output included at the last*

**Testing in TACC Machine:**

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 are 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.

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

*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*

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

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

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

*Concurrency correctness PASSED!!!*

*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**