CS5300-Parallel and concurrent programming

Fall-2023

Comparing different Queue implementations

Programming assignment-2

Name-Souvik Sarkar

(CS23MTECH02001)

Objective:

The goal of this assignment is to implement different queue implementations studied in the class and compare their performance. Implement these algorithms in C++.

Program Design :

i) Coarse grain lockbase Queue CLQ :

1. LockBasedQueue is a template class for creating a lock-based queue with a specified capacity.

2. The enq method is used to enqueue an element of type T into the queue.
    a. It first locks the mutex to ensure exclusive access to the shared data structures.
    b. Then, it checks if the queue is full. If it is, you can choose to either throw an exception or handle the situation in a way that makes sense for your application.
    c. If the queue is not full, it inserts the element at the tail position and increments the tail pointer.
    d. Finally, it unlocks the mutex to allow other threads to access the queue.

3. The deq method is used to dequeue an element from the queue.
    a. Similar to enq, it locks the mutex to ensure exclusive access.
    b. It checks if the queue is empty. If it is, you can choose to either throw an exception or handle the situation accordingly.
    c. If the queue is not empty, it retrieves the element at the head position, increments the head pointer, and returns the dequeued element.
    d. Finally, it unlocks the mutex to allow other threads to access the queue.

ii) Non-Locking Queue NLQ :

1. HWQueue is a template class representing a lock-free queue with a fixed capacity (CAPACITY).
2. The constructor initializes the queue by setting tail to zero and initializing all items in the queue with nullptr. This is done to prepare the circular buffer for future enqueue and dequeue operations.
3. The enq method is used to enqueue an element (T* x) into the queue.
    a. It atomically increments the tail counter and retrieves the previous value using fetch_add with relaxed memory ordering.
    b. Then, it stores the element at the corresponding position in the circular buffer.

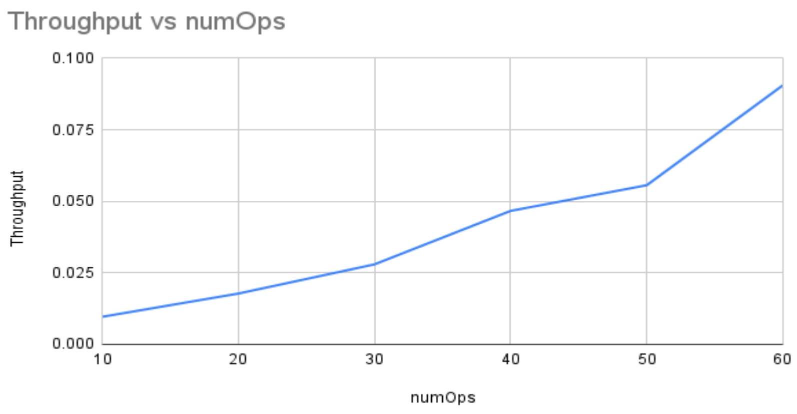
4. The deq method is used to dequeue an element from the queue.

    a. It continuously loops to check for available elements in the queue.
    b. It retrieves the current value of tail with relaxed memory ordering to determine the range of elements to check.
    c. It iterates through the circular buffer, atomically exchanging each element with nullptr.
    d. If a non-null element is found, it is returned as the result of the dequeue operation.
    e. This loop continues until a valid element is found or the thread is preempted by another thread.
    f. If the queue is empty (all elements are nullptr), it returns nullptr.

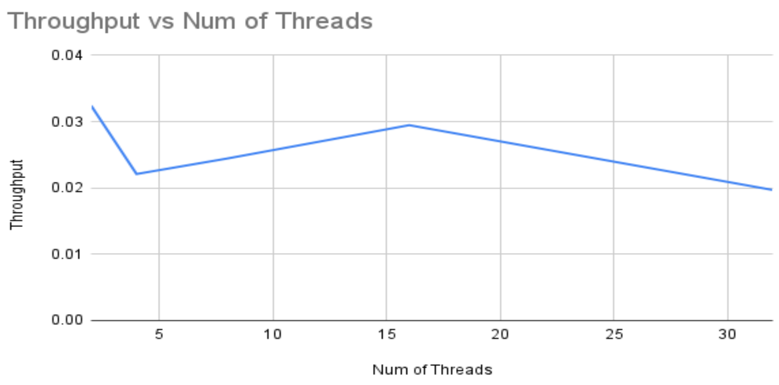
Note: This code assumes that memory management for elements (e.g., allocating and deallocating memory) is handled externally and that the elements themselves are pointers (T*).

<u>Performance Of CLQ</u> :

<u>numOps vs Throughput graph</u> :



<u>Thread numbers vs Throughput graph</u> :



<u>Observation</u> :

In the CLQ queue implementation we are locking the queue before any operation this makes the Queue access mutual exclusive. This ensures the consistency of the queue but it is affecting the throughput and overall performance of the program.

Performance Of NLQ :

In the Non-locking implementation the Deq() function is not lockfree it stuck in infinite loop if there is no enqueue operation. Even if there are enq operations but the numbers of enqueue operation is less then deq operations then also our program stuck in a infinite loop and never terminate .For this type of non-deterministic behaviour we are unable to plot the performance graph of NLQ queue implementation.