Comparing Different Queue Implementations
CO21BTECH11004

As discussed in class, NLQ implementation is given in the book; it would get stuck if all thread call deq and queue is empty.
randLT is set to 0.7, so more enqueue operations than dequeue.
Run NLQ again if it stuck, as all threads would be doing dequeue.

Implementation: -

- Global variables numOps, randLT and lambda are created.
- Global array threadEnqTime, threadDeqTime, and thrTime are dynamically allocated in main of size as number of threads.
- Global queue is created in which treads will call enq and deq operations.
- C++ threads are used, created n threads which call testThread function.
- Time is measured in microseconds using chrono library.
- Random number to decide enq/deq operation is generated using uniform_real_distribution, while that for sleep is generated using exponential_districution using random library.
-

CLQ1:-

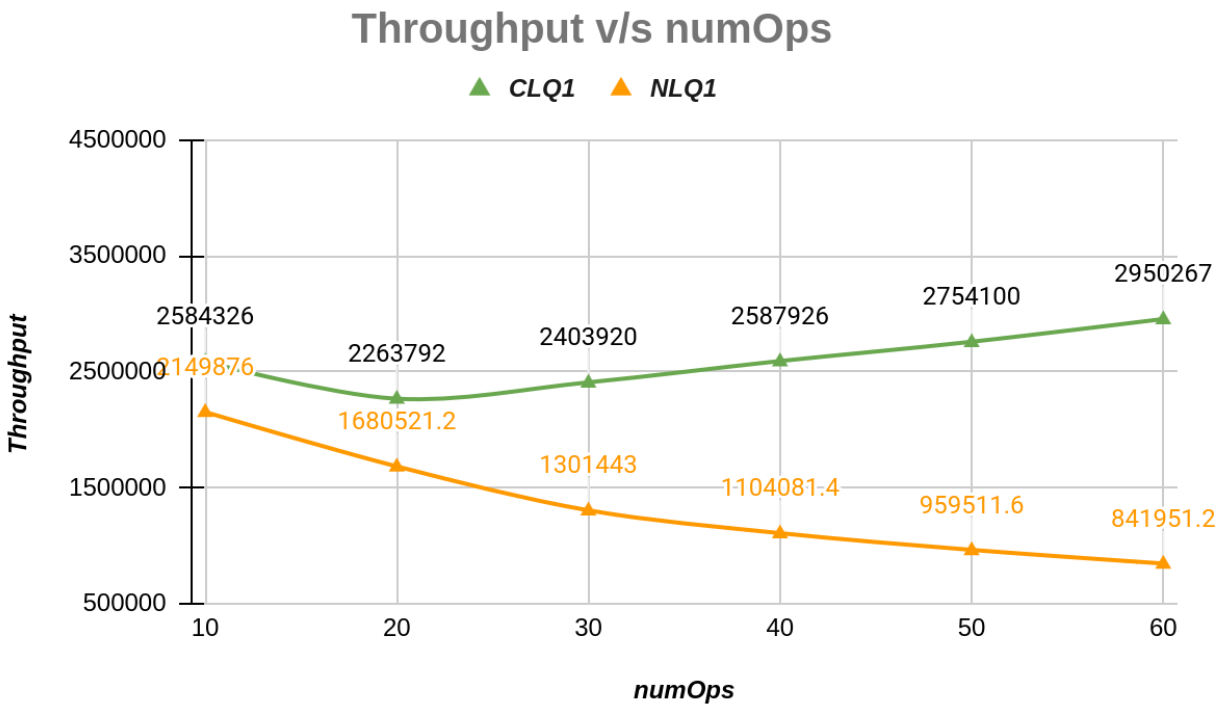
- LockBasedQueue class is implemented as per book code.
- Mutex lock is used.

NLQ1: -

- HWQueue class is implemented as per book code.
- atomic (int) is used for the tail and items array.
- fetch_add(1) is used instead of getAndIncreament.
- tail.load() is used for getting the tail value.
- exchange(-1) is used instead of getAndSet(Null).

Plots

Throughput vs numOps (number of operations)



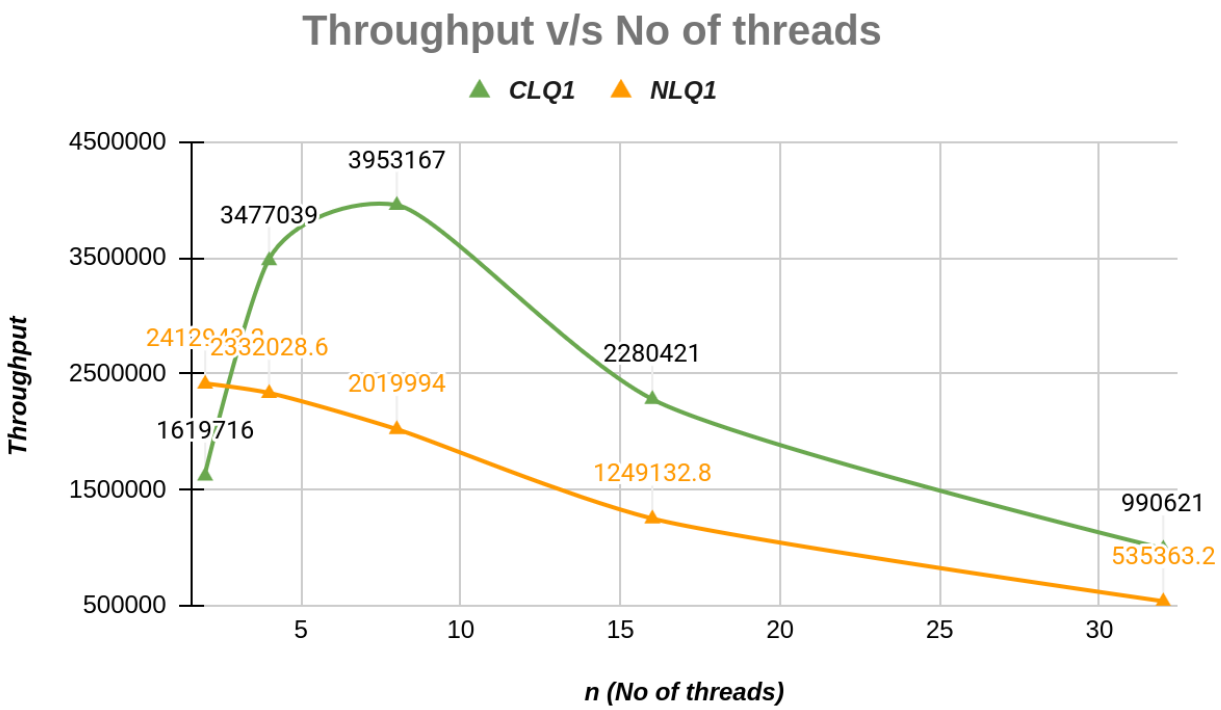
- For NLQ, throughput decreases as the number of operations increases - (2.15e6 - 8.4e5)
    - deq() operation is not constant time. When a thread calls deq(), it searches the entire items array and terminates when it finds an element.
    - Size of items increases as numOps increases, so search time increases, so on average, more time is spent on operations, so throughput decreases.
- For CLQ, we can say throughput is almost constant - (2.5e6 - 2.9e6)
    - Both enq(x) and deq() are constant time operations, so there is not a significant difference in average time for one operation, so average throughput almost constant.
- CLQ is performing better than NLQ, as the time complexity of deq() increases in NLQ as queue size increases with the number of operations. Also, the time thread waiting for the lock is very less as enq/deq for CLQ are constant time methods as compared to deq in NLQ.

Data used for plotting graph

CLQ1 numOps	Throughput in each iteration					Throughput Average
	1st	2nd	3rd	4th	5th	
10	2807017	3018867	2580645	2162162	2352941	2584326
20	1963190	2689075	2253521	1893491	2519685	2263792
30	2211981	1811320	2424242	3018867	2553191	2403920
40	2735042	2976744	2442748	2140468	2644628	2587926
50	3041825	2666666	3137254	2507836	2416918	2754100
60	2917933	2848664	2807017	3009404	3168316	2950267

NLQ1 numOps	Throughput in each iteration					Throughput Average
	1st	2nd	3rd	4th	5th	
10	2025316	2318840	2162162	2191780	2051282	2149876
20	1787709	1684210	1434977	1624365	1871345	1680521.2
30	1212121	1151079	1293800	1340782	1509433	1301443
40	1176470	1020733	1099656	1196261	1027287	1104081.4
50	849256	947867	924855	1085481	990099	959511.6
60	817021	798668	854853	931134	808080	841951.2

Throughput vs No of threads



- For NLQ, as number of thread increases so more enq/deq operations in total increases. So, dequeue has to traverse the items list more to search for non-null values. On average, dequeue time increases, so throughput decreases.
- For CLQ, throughput first increases and then decreases.
    - Here for n=2, 2*30 = 60 enq/deq operations are performed, which are very few, upto n=8, time does not increase significantly, but after n=8, linear increase in time is there while the number of steps increases by a factor of 2 in both cases.
- CLQ has higher throughput than NLQ as the dequeu operation of NLQ is polynomial time, while for CLQ that is constant. Also, the time thread waiting for the lock is very less as enq/deq for CLQ are constant time methods.

Data used for plotting graph

CLQ1 n	Throughput in each iteration					Throughput Average
	1st	2nd	3rd	4th	5th	
2	1428571	1764705	1666666	1875000	1363636	1619716
4	3157894	3750000	3076923	3870967	3529411	3477039
8	4285714	4137931	3692307	3582089	4067796	3953167
16	2513089	2133333	2264150	2042553	24489799	2280421
32	1160822	917782	1070234	885608	918660	990621

NLQ1 n	Throughput in each iteration					Throughput Average
	1st	2nd	3rd	4th	5th	
2	2068965	2222222	2857142	2608695	2307692	2412943.2
4	1428571	2608695	2142857	2926829	2553191	2332028.6
8	2285714	1889763	2142857	1846153	1935483	2019994
16	1359773	1156626	1325966	1140142	1263157	1249132.8
32	513643	534223	472906	598877	557167	535363.2

CLQ2 Implementation: -

- Since lock is used so there is no overlap of enq and deq operations.
- So if enq will start after acquiring the lock, so if enq has acquired the lock and called deq , then it is guaranteed that deq will return the enq value.
- Due to the presence of lock, all these operations take effect sequentially, so any method calls a before b would be consistent as per behaviour of the queue.
- front and rear can be implemented using locks same way enq/deq are implemented.
- front() :- acquire the lock, get the element at the head, release lock, and return the element.
- rear() :- acquire the lock, get the element at the tail, release lock, and return the element.