Parallel Data Structures

Queue and Linked List

Daniel Rivera Ruiz (N10385937)
Fatima Mushtaq (N15760010)

Professor Mohamed Zahran
CSCI-GA.3033-016 Multico Processors
NYU - Courant Institute of Mathematics

Problem Definition

- Like other programming languages, the goal is to support concurrent thread-safe data-structures in **OpenMP.**
- Provide Parallel Implementation for two data structures: **Queue** and **Linked List.**

Requirements:

1. **Correctness (linearizability):** the responses received in every concurrent history are equivalent to those of a legal sequential history and their order is consistent with the real-time order.
2. **Scalability:** at least some of the operations defined for each data structure have better performance as the number of threads increases.

Experiments

For both data structure, developed two implementations: **lock-based** and **lock-free.**

- **Lock-based** implementation uses *OpenMP* lock functions: *test, set, unset,* etc.
- **Lock-free** implementation uses C++ libraries: *<atomic>*, *pthreads or OpenMP.*

Concurrent Queue Implementation supports **6 APIs**:

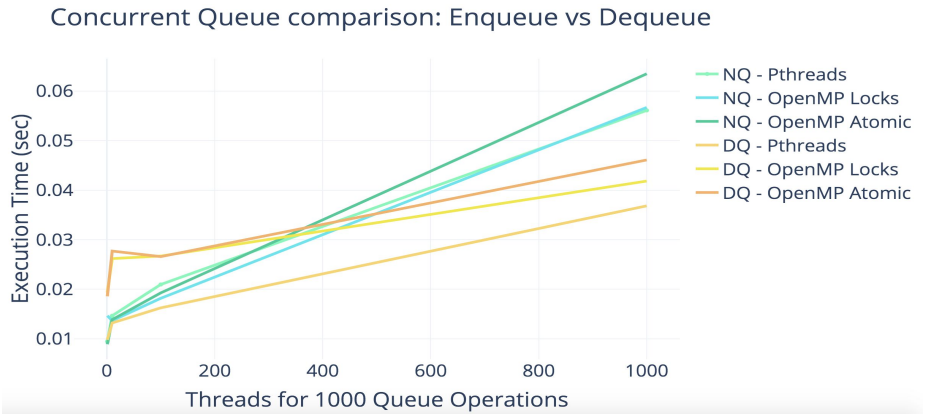
- *EnQueue(T), T DeQueue(), T head(), T tail(), isEmpty(), isFull()*
- *Lock free Implementation is done in a way that it uses **one Atomic Stmnt only i.e.** atomic capture.*

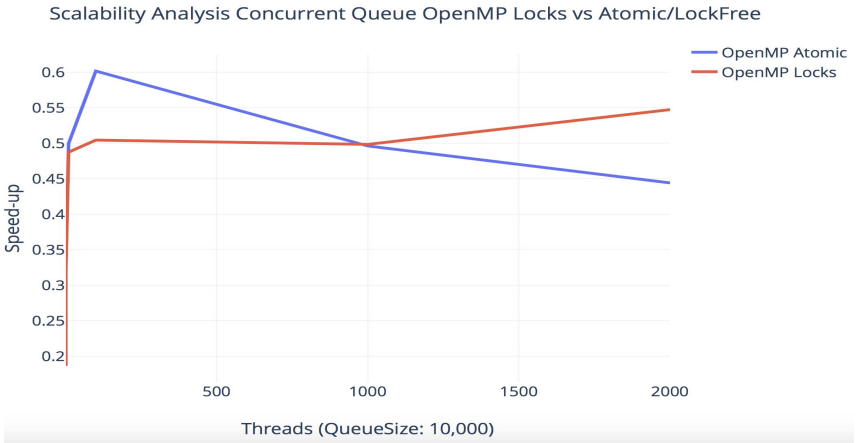
Concurrent Linked List Implementation supports **15 APIs**, mainly:

- *add(index, T), remove(index), get(index), set(index, T), indexOf(T), contains(T)*

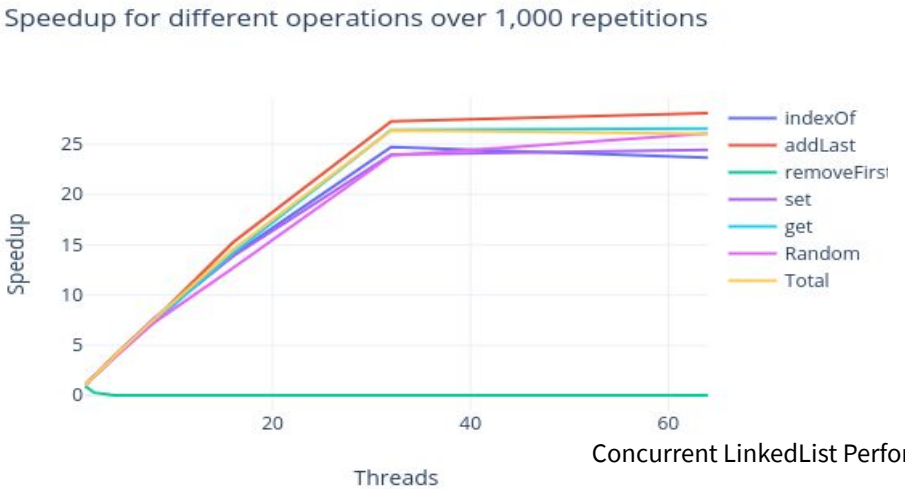
We performed several experiments for each implementation, variating the **number of threads** and the **type and number of operations** performed. The main metric we used to evaluate performance is execution time as measured with **omp_get_wtime()**.

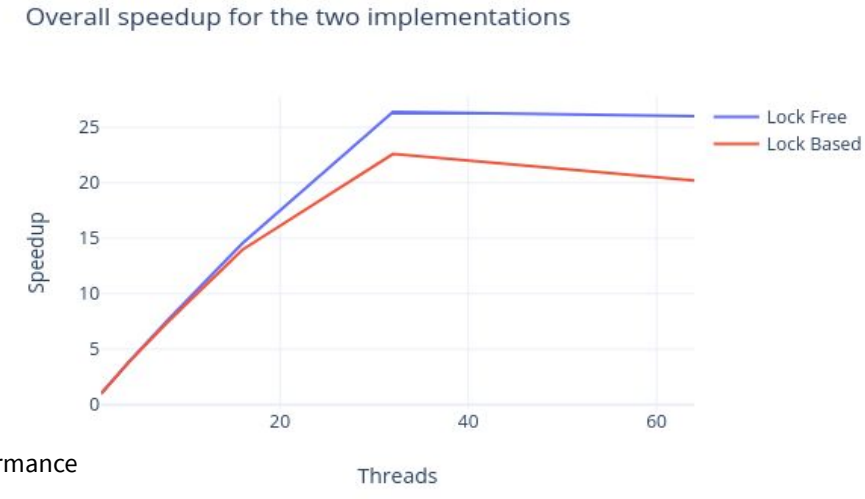
Results





Concurrent Queue Performance





Concurrent LinkedList Performance

Conclusions

Queue:

- Pthreads/Mutex has shown to be more efficient than OpenMP Implementation.
- Lock-Free approach is complex. It showed less performance gain in OpenMP.
- Concurrent queue implementation in OpenMP scales with problem size and number of threads.

Linked List:

- Operations that try to modify the list at the same location every time (e.g. addFirst(), deleteFirst(), etc.) do not benefit from parallelism.
- The most challenging operation in the parallel implementation of a Linked List is deletion, since it requires two atomic operations to ensure correctness.