Concurrent Data Structures – Lab Assignment 2

Parosh Aziz Abdulla
Sarbojit Das

December 10, 2022

Deadline: 2022–12–23 23.59.

Please, submit your source code, instructions to run them, and the tables/curves with explanations.

Task 1 Fine-grained synchronization Implement a concurrent list-based set with Fine-Grained synchronization in a file named "FineList.cpp".

Task 2 Optimistic synchronization Implement a concurrent list-based set with Optimistic synchronization in a file named "OptimisticList.cpp".

Task 3 Lazy synchronization Implement a concurrent list-based set with Lazy synchronization in a file named "LazyList.cpp".

Task 4 Experiment Performs the following experiments:

- For each value $i = 10, 50$, and 90, create a test case having 500 operations such that i% of the operations are `ctn()`. From the remaining set, 90% should be `add`, and 10% should `rmv()`. For instance, for $i = 60$, we have 60% `ctn()`, 36% `add()`, and 4% `rmv()`. The type of the operations are selected randomly, and the input values are selected randomly from the set $\{0, 1, \ldots, 7\}$.
- Repeat the following experiment for each i test case. Create n worker threads, where $n = 2, 4, 8, 16, 32$ and each worker thread performs the operations from the same test case. Measure the total running time of n threads.
- Repeat the previous experiments with input values $\{0, 1, \ldots, 1023\}$.

Depict the results in tables or (preferably) curves, where the x-axis is the number of threads, and the y-axis is the running time. Explain the tables or curves.

Task 5 Multiset A multiset is a generalization of a set where a given element may occur multiple times in the multiset. An example of a multiset (over the set of integers) is [2; 2; 2; 3; 7; 7]. Here, the number of occurrences of 2, 3, and 7 is 3, 1, and 7, respectively. A multiset can also be viewed as a special case of a stack or a queue, where the order of the elements is not relevant. For this assignment you are tasked to implement a concurrent multiset that supports the following functions:

- `add(x)`: Adds element x to the set and returns true
- `rmv(x)`: If possible, removes (an instance of) element x from the set and returns true, otherwise false
- `cnt(x)`: Returns the multiplicity of element x(how many instances of x there are in the multiset)

Implement a concurrent multiset using Fine-Grained synchronization. Create 16 worker threads, each does 100 operations on FineList. Mix various operations `add()`, `rmv()`, and `cnt()`. Identify linearization policy and check if it is consistent with non-concurrent multiset ADT. You can create a monitor function exactly like the first assignment. Create a shared sequence of operations. Whenever a worker does some operation, put it in the shared sequence at the linearization point. The monitor reads the sequence and do the operations on non-concurrent multiset. Here you can find c++ multiset: std::multiset.