CSE 375/475 - Fall 2025
Assignment 2: Concurrent Open-Addressed Hash Table
Due date: Sunday November 9

This assignment aims to implement three versions of an open-addressed hash set. To read more about open-addressed hash set design, you can find an introduction to that in Chapter 13.4 of the textbook "The Art of Multiprocessor Programming" and many other books or online resources. The set implementation has to provide the following APIs as described in the textbook: 'add', 'remove', and 'contains'. In addition to that, the set should provide a non-thread safe function (named 'size') that counts the number of elements in the hash set, and a non-thread safe function (named 'populate') that initializes the data structure with random elements (feel free to change the size to test different configurations). The set must allow applications to use any generic (comparable) data type. Implementations that account for only integers are accepted with a penalty of 5 points.

The following three versions of an open-addressed set are required:

- A sequential version, as specified in Section 13.4.1.
- A concurrent version with a fixed number of locks, as specified in Sections 13.4.2 and 13.4.3.
- A concurrent version that uses the Transactional Memory abstraction as provided by GCC to convert the sequential version into concurrent.

In order to test the performance of the three versions, it is required to develop a simple application that spawns a pre-defined number of threads that operate on the implemented open-addressed hash set of integer elements. Each of these threads executes a pre-defined number of operations on the data structure with a pre-defined distribution (e.g., 1M operations, 80% contains, 10% insert; 10% remove). Elements to be inserted, removed, or searched by these operations are random.

The outcome of each operation should be recorded so that the final size of the data structure at the end of the execution can be computed. When all threads are joined, the function ‘size’ should be invoked and it should match the value of the computed size.

When you perform your test, do not forget to populate your data structure first and then start benchmarking your application.

The goal of the assignment is to contrast the performance of the two concurrent implementations (with the sequential version as a baseline to measure scalability). It is required to produce enough plots that show performance exploring different configurations.

It is a requirement to use the machines available at sunlab.cse.lehigh.edu. An evaluation performed on your own laptop will not be considered enough, unless a specific authorization has been granted by me.

Important notes about submission and grading

- The developing language is C++.
- TM is implemented in GCC starting from version 4.7. Use the latest version of GCC available in Sunlab.
- All the findings (successful and unsuccessful) should be recorded in a document to be submitted along with the implementations' source code.
- The submission should be done through coursesite.lehigh.edu and should consist of:
    - The source code of your assignment.
    - A document containing the performance plots and the findings you consider worth to be shared with me.
- Assignments will be graded during a one-on-one meeting with each student. You must be able to explain everything in your submission, including design attempts, plots, and analysis.

Good luck!