**Introduction**

This project consisted in studying, understanding, implementing, and testing a scalable, correct time-stamped stack. The main idea of this data structure is that the order of the nodes in the stack is only important if the insert methods are performed sequentially. This data structure relies in creating timestamps and associating them with each respective node to avoid ordering. To maintain a LIFO structure, items are popped with respect to timestamps rather than position in the stack.

**Implementation**

The timestamped stack consists of 3 classes: the stack, which holds the push and pop operations, the TS buffer, which is the underlying data structure used to implement the timestamped stack, and the SP buffer, which is a singly-linked list used by the TS stack.

The stack class holds a single TS Buffer and the push and pop methods. The TS buffer class consists of a collection of single-producer multiple-consumer buffers, or SP buffers. Each thread is then associated with a SP buffer to insert elements, but it’s allowed to remove elements from any SP buffer as it searches for the one with the closest timestamp.

Timestamping

There are different ways to implement a method that generates timestamps to later associate them with items. The reference document provided different ideas on how to approach this problem and how each compared to one another. Some of these included:

* Atomic timestamping: using a global counter and perform fetch-and-increment instructions.
* Hardware timestamping: using the current value of the machine timer.
* Stutter timestamping: using thread-local counters that are then synchronized using Lamport’s algorithm.
* Interval timestamping: returning an interval of timestamps generated by one of the algorithms above.

**Advantages**

The TS (timestamped) stack was tested alongside the Treiber stack and the EB (exponential back-off) stack in two machines different machines, a 40-core and a 64-core machine. According to the results, when the thread count exceeded 16, the TS stack outperformed the Treiber stack and on both machines, it outperformed the EB stack by a factor of 2.

Timestamping has been used before in other data structures (Gorelik and Hendler’ AFC queue). Aside from the obvious differences, a main advantage over their implementation is that the AFC queue is blocking because it uses a merger to combine and consolidate the timestamps in order. The LCRQ and the SP queue both index the elements (similarly to using timestamps) using an atomic counter. The reason the TS stack timestamp approach is advantageous is because the queues’ performance suffers when the dequeue counter becomes higher than the enqueue counter.

**Personal Observations**

All the group members encountered very similar obstacles with this project. The main one is that all members are more familiar and prefer using Java as a programming language than C++ and becoming comfortable and proficient with C++ took a period of adjustment. Another point of interest is that the data structure, or at least the design document, was first introduced in 2014 and there isn’t any meaning supplemental information available for this project.