CS5300 - Parallel Concurrent Programming Fall 2023
Programming Assignment 3: Implementing Multi Reader Multi Writer Register

Souvik Sarkar(CS23MTECH02001)

1 Detailed Program Design Explanation

1.1 Classes and Structures

1.1.1 `StampedValue<T>` Class:

- **Purpose:** Represents a timestamped value.
- **Functions:**
  - `StampedValue(T init)`: Constructor initializing a StampedValue with a given value.
  - `StampedValue(long stamp1, T value1)`: Constructor initializing a StampedValue with a timestamp and value.
  - `static StampedValue<T> max(StampedValue x, StampedValue y)`: Static method to find maximum StampedValue based on timestamp.
- **Data Members:**
  - `long stamp`: Timestamp associated with the value.
  - `T value`: The stored value.
- **Usage:** Used to store values with associated timestamps for comparison and synchronization purposes.

1.1.2 `AtomicMRMWRegister<T>` Class:

- **Purpose:** Manages a vector of `StampedValue<T>` objects representing atomic MRSW registers.
- **Functions:**
  - `AtomicMRMWRegister(int capacity, T init)`: Constructor initializing the register with a given capacity and initial value.
  - `void write(T value, int ThreadID)`: Writes a value to the register with a specific thread ID.
  - `T read()`: Reads a value from the register.
- **Data Members:**
  - `std::vector<StampedValue<T>> a_table`: Vector storing atomic MRSW registers.
- **Usage:** Ensures synchronization among threads accessing the register by implementing read and write operations.

1.1.3 `printTimestamp` Function:

- **Purpose:** Prints the current timestamp in the format HH:MM:SS.
- **Parameters:** `std::chrono::high_resolution_clock::time_point timestamp`, `FILE* file`.
- **Usage:** Utilized for logging timestamps in the log file.

2 Performance comparision

2.1 Impact of average time with increasing Capacity:

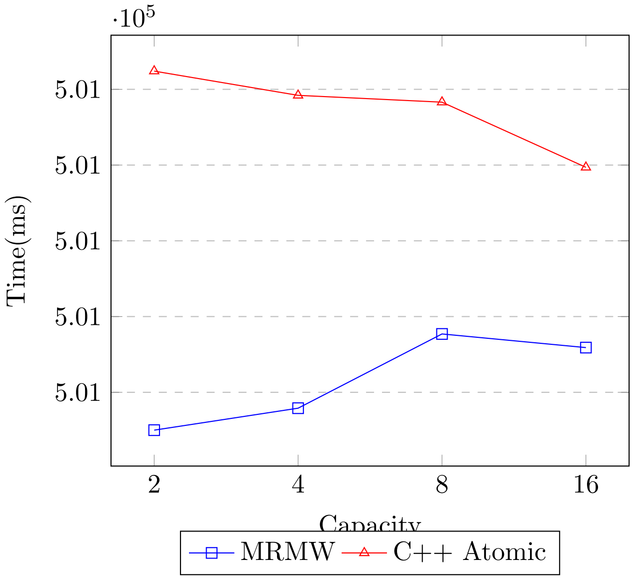

Figure 1: Comparison of MRMW and C++ Atomic values based on capacity.

2.2 Impact of average time with increasing numOps:

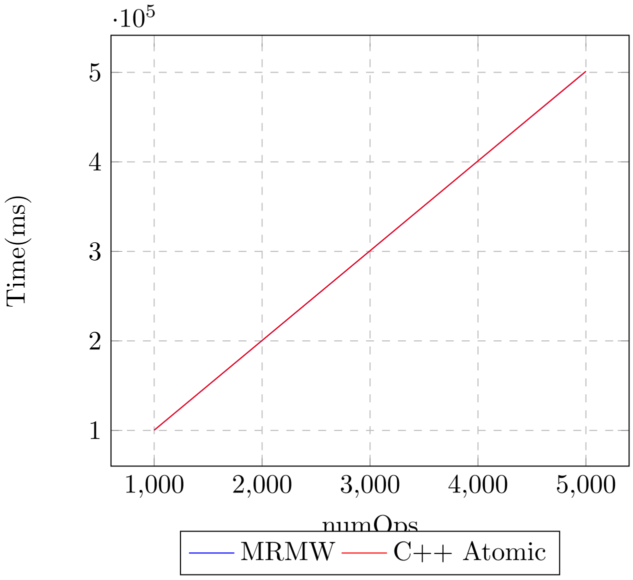

Figure 2: Comparison of MRMW and C++ Atomic values based on numOps.

3 Observation :

AtomicMRMWRegister, being wait-free, ensures progress for every thread but comes with complexity due to its intricate implementation. In contrast, using C++'s Atomic class offers simpler and more efficient concurrent operations, making it perform better in scenarios where wait-free guarantees are not essential.