Thread Local Summation: Using an Atomic Variable with Sequential Consistency

This lesson explains the solution for calculating the sum of a vector problem using an atomic variable with sequential consistency in C++.

we'll cover the following ^

Using an Atomic Variable with Sequential Consistency

Using an Atomic Variable with Sequential Consistency

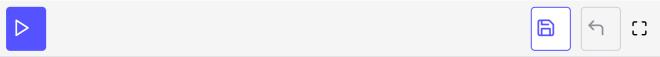
Let's replace the non-atomic global summation variable sum with an atomic.

Let's see it in action:

```
// localVariableAtomic.cpp

#include <chrono>
#include <iostream>
#include <mutex>
#include <random>
#include <thread>
#include <utility>
#include <vector>
#include <atomic>
```

```
constexpr long long size = 100000000;
constexpr long long fir = 25000000;
constexpr long long sec = 50000000;
constexpr long long thi = 75000000;
constexpr long long fou = 100000000;
std::mutex myMutex;
std::atomic<unsigned long long> sum = {};
void sumUp(std::atomic<unsigned long long>& sum, const std::vector<int>& val,
           unsigned long long beg, unsigned long long end){
      unsigned int long long tmpSum{};
    for (auto i = beg; i < end; ++i){
        tmpSum += val[i];
    sum+= tmpSum;
}
int main(){
  std::cout << std::endl;</pre>
  std::vector<int> randValues;
  randValues.reserve(size);
  std::mt19937 engine;
  std::uniform int distribution<> uniformDist(1,10);
  for (long long i = 0; i < size; ++i)
      randValues.push_back(uniformDist(engine));
  const auto sta = std::chrono::steady_clock::now();
  std::thread t1(sumUp, std::ref(sum), std::ref(randValues), 0, fir);
  std::thread t2(sumUp, std::ref(sum), std::ref(randValues), fir, sec);
  std::thread t3(sumUp, std::ref(sum), std::ref(randValues), sec, thi);
  std::thread t4(sumUp, std::ref(sum), std::ref(randValues), thi, fou);
  t1.join();
  t2.join();
  t3.join();
  t4.join();
  std::chrono::duration<double> dur= std::chrono::steady_clock::now() - sta;
  std::cout << "Time for addition " << dur.count()</pre>
            << " seconds" << std::endl;
  std::cout << "Result: " << sum << std::endl;</pre>
  std::cout << std::endl;</pre>
}
```



In the next lesson, we'll be discussing thread local summation using an atomic variable with relaxed semantic.