

Multithreaded Summation: Using Atomic Variable

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

Now, the summation variable `sum` is an atomic; that means I don't need the `std::lock_guard` anymore. Here is the modified `sumUp` function.

```
// synchronisationWithAtomic.cpp

...

void sumUp(std::atomic<unsigned long long>& sum, const std::vector<int>& val,
           unsigned long long beg, unsigned long long end){
    for (auto it = beg; it < end; ++it){
        sum += val[it];
    }
}

...
```

Let's see the above fragment of code in action:

```
// synchronisationWithAtomic.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){
    for (auto it = beg; it < end; ++it){
        sum += val[it];
    }
}
```

```

    }
}

int main(){

    std::cout << std::endl;

    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()
        << " seconds" << std::endl;
    std::cout << "Result: " << sum << std::endl;

    std::cout << std::endl;
}

```



The synchronization with `std::lock_guard` is more than twice as fast as the atomic version.

In addition to using the `+=` operator on an atomic, you can use the `fetch_add` method. Let's try it out.