Calculate Sum of a Vector: Conclusion

This lesson concludes all the methods used to solve the problem of calculating the sum of a vector in C++.

WE'LL COVER THE FOLLOWING

Let's conclude what we learned from this chapter:

Single Threaded

The range-based for loop and the STL algorithm `std::accumulate` are in the same performance range. This observation holds for the most optimized version. In the optimized version, the compiler uses the optimized version vectorized SIMD instruction (SSE or AVX) for the summation case; therefore, the loop counter will be increased by 2 (SSE) or 4 (AVX).

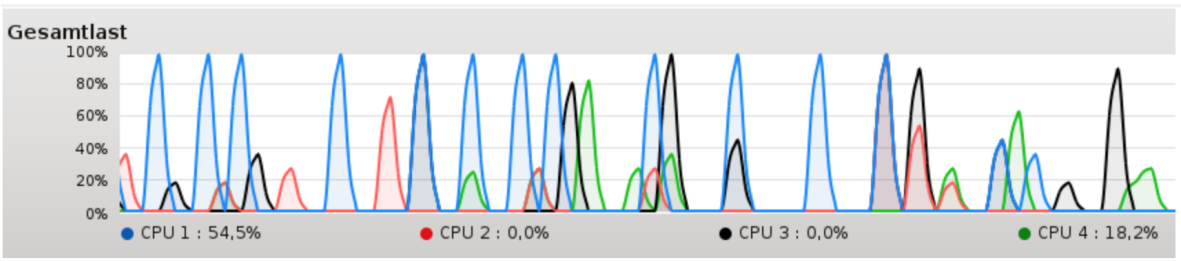
Multithreading with a Shared Variable

The usage of a shared variable for the summation variable makes one point clear: synchronization is very expensive and should be avoided as much as possible. Although I used an atomic variable and even broke the sequential consistency, the four threads are 100 times slower than one thread. From a performance perspective, minimizing expensive synchronization has to be our first goal.

Thread-local Summation

Thread-local Summation

The thread-local summation is only two times faster than the single-threaded range-based for loop or `std::accumulate`; that holds, even though each of the four threads can work independently. That surprised me because I was expecting a nearly fourfold improvement, and it surprised me even more that my four cores were not fully utilized.



The reason is simple: the cores can't get the data fast enough from the memory. To that point, the execution is memory bound - i.e. it slows down the cores.