

async: Concurrent Calculation

This lesson gives an overview of concurrent calculation used with `std::async` in C++ for multithreading.

The calculation of the scalar product can be spread across four asynchronous function calls.

```
// dotProductAsync.cpp

#include <iostream>
#include <future>
#include <random>
#include <vector>
#include <numeric>

using namespace std;

static const int NUM= 100000000;

long long getDotProduct(vector<int>& v, vector<int>& w){

    auto vSize = v.size();

    auto future1 = async([&]{
        return inner_product(&v[0], &v[vSize/4], &w[0], 0LL);
    });

    auto future2 = async([&]{
        return inner_product(&v[vSize/4], &v[vSize/2], &w[vSize/4], 0LL);
    });

    auto future3 = async([&]{
        return inner_product(&v[vSize/2], &v[vSize* 3/4], &w[vSize/2], 0LL);
    });

    auto future4 = async([&]{
        return inner_product(&v[vSize * 3/4], &v[vSize], &w[vSize * 3/4], 0LL);
    });

    return future1.get() + future2.get() + future3.get() + future4.get();
}

int main(){

    cout << endl;

    random_device seed;

    // generator
```

```

// generator
mt19937 engine(seed());

// distribution
uniform_int_distribution<int> dist(0, 100);

// fill the vectors
vector<int> v, w;
v.reserve(NUM);
w.reserve(NUM);
for (int i=0; i< NUM; ++i){
    v.push_back(dist(engine));
    w.push_back(dist(engine));
}

cout << "getDotProduct(v, w): " << getDotProduct(v, w) << endl;

cout << endl;
}

```



The program uses the functionality of the random and time libraries - both libraries are part of C++11. The two vectors `v` and `w` are created and filled with random numbers (lines 50 - 56). Each of the vectors (lines 53 - 56) gets one hundred million elements. `dist(engine)` in lines 54 and 55 generates the random numbers, which are uniformly distributed in the range 0 to 100. The calculation of the scalar product takes place in `getDotProduct` (lines 13 - 34). Internally, `std::async` uses the standard template library algorithm `std::inner_product`. The return statement sums up the results of the futures. `std::packaged_task` is also usually used to perform a concurrent computation.