- Solution

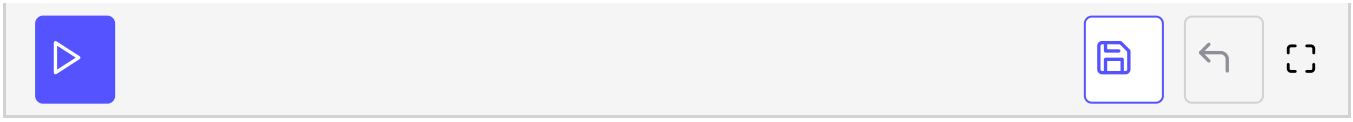
The solution to the task iof the previous exercise will be explained in this lesson.

WE'LL COVER THE FOLLOWING

Solution

```
// dotProductAsync.cpp

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

static const int NUM= 100000000;

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

  auto future1= std::async([&]{return std::inner_product(&v[0], &v[v.size()/4], &w[0], 0LL);}
  auto future2= std::async([&]{return std::inner_product(&v[v.size()/4], &v[v.size()/2], &w[v
  auto future3= std::async([&]{return std::inner_product(&v[v.size()/2], &v[v.size()*3/4], &w
  auto future4= std::async([&]{return std::inner_product(&v[v.size()*3/4], &v[v.size()], &w[v

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


int main(){

  std::cout << std::endl;

  // get NUM random numbers from 0 .. 100
  std::random_device seed;

  // generator
  std::mt19937 engine(seed());

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

```
  // fill the vectors
  std::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));
  }

  // measure the execution time
  std::chrono::system_clock::time_point start = std::chrono::system_clock::now();
  std::cout << "getDotProduct(v, w): " << getDotProduct(v, w) << std::endl;
  std::chrono::duration<double> dur   = std::chrono::system_clock::now() - start;
  std::cout << "Parallel Execution: "<< dur.count() << std::endl;

  std::cout << std::endl;

}
```



Explanation

- `std::async` is quite convenient to put a bigger computation job on more shoulders. The calculation of the scalar product is done in the program with four asynchronous function calls.
- The calculation of the scalar product takes place in `getDotProduct` (lines 11 - 19). Internally, `std::async` uses the standard template library algorithm `std::inner_product`. The return statement sums up the results of the futures.
- The performance improvements may vary depending on your platform, but you can expect a two to four times performance improvement to the single-threaded version.

For further information, see std::async

In the next lesson, we will show you how to parallelize a big compute job by using `std::packaged_task`.