- Exercise

In this exercise, you will design asynchronous functions and compare the performance of single and multithreaded programs.

WE'LL COVER THE FOLLOWING

Task 1

The calculation of the scalar product in the program below is quite easy to parallelize. Take a look at the explanation below to better understand the code.

Launch four asynchronous functions to calculate the scalar product.

```
// dotProduct.cpp

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

static const int NUM= 100000000;

long long getDotProduct(std::vector<int>& v, std::vector<int>& w){
  return std::inner_product(v.begin(), v.end(),
                           w.begin(),
                           0LL);
}

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 << "Sequential Execution: "<< dur.count() << std::endl;

  std::cout << std::endl;

}
```

Explanation

- The program uses the functionality of the random and time library. Both libraries are part of C++11.
- The two vectors `v` and `w` are created and filled with random numbers in lines 31 - 37.
- Each of the vectors gets (lines 34 - 37) one hundred million elements. `dist(engine)` in lines 35 and 36 generated the random numbers, which are uniformly distributed in the range from 0 to 100.
- The current calculation of the scalar product takes place in the function `getDotProduct` (lines 11 - 15). We have explicitly used the standard template library algorithm `std::inner_product` for the computation.
- The return statement sums up the results of the futures.

Try It Out!

Compare the execution time of the single and multithreaded programs. When you make a performance test, go for maximum optimization. Consider the following question: are the cores fully utilized? If not, why?

You can find the solution to this exercise in the next lesson.