- Exercise

In this lesson, we'll solve an exercise.

WE'LL COVER THE FOLLOWING

- Problem statement

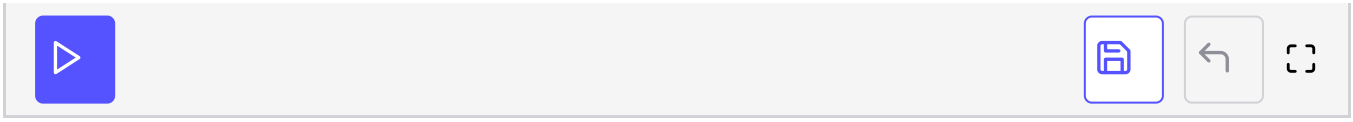
Problem statement

The code is quite easy to parallelize.

- Launch four asynchronous functions to calculate the scalar product.
- Compare the execution time of the single and multithreaded program.
- Are the cores fully utilized?

```
#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;

}
```



The solution will be explained in the next lesson.