# Thread Local Summation: Using Tasks

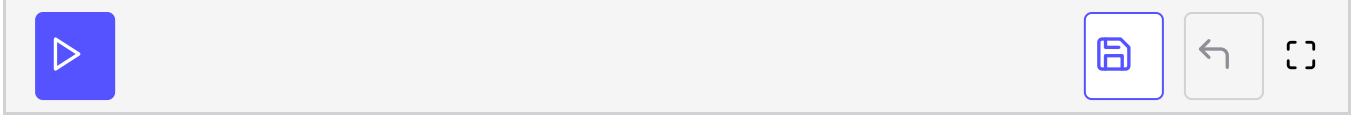
This lesson explains the solution for calculating the sum of a vector problem using tasks in C++.

Using tasks, we can do the whole job without synchronization. Each partial summation is performed in a separate thread and the final summation takes place in the main thread.

Here is the program:

```
// tasksSummation.cpp

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

constexpr long long size = 100000000;

constexpr long long fir =  25000000;
constexpr long long sec =  50000000;
constexpr long long thi =  75000000;
constexpr long long fou = 100000000;

void sumUp(std::promise<unsigned long long>&& prom, const std::vector<int>& val,
           unsigned long long beg, unsigned long long end){
    unsigned long long sum={};
    for (auto i = beg; i < end; ++i){
        sum += val[i];
    }
    prom.set_value(sum);
}

int main(){

  std::cout << std::endl;

  std::vector<int> randValues;
  randValues.reserve(size);

  std::mt19937 engine;
  std::uniform_int_distribution<> uniformDist(1,10);
  for (long long i = 0; i < size; ++i)
      randValues.push_back(uniformDist(engine));
```

```
  std::promise<unsigned long long> prom1;
  std::promise<unsigned long long> prom2;
  std::promise<unsigned long long> prom3;

  std::promise<unsigned long long> prom4;

  auto fut1= prom1.get_future();
  auto fut2= prom2.get_future();
  auto fut3= prom3.get_future();
  auto fut4= prom4.get_future();

  const auto sta = std::chrono::system_clock::now();

  std::thread t1(sumUp, std::move(prom1), std::ref(randValues), 0, fir);
  std::thread t2(sumUp, std::move(prom2), std::ref(randValues), fir, sec);
  std::thread t3(sumUp, std::move(prom3), std::ref(randValues), sec, thi);
  std::thread t4(sumUp, std::move(prom4), std::ref(randValues), thi, fou);

  auto sum= fut1.get() + fut2.get() + fut3.get() + fut4.get();

  std::chrono::duration<double> dur= std::chrono::system_clock::now() - sta;
  std::cout << "Time for addition " << dur.count()
            << " seconds" << std::endl;
  std::cout << "Result: " << sum << std::endl;

  t1.join();
  t2.join();
  t3.join();
  t4.join();

  std::cout << std::endl;

}
```



In lines 39 - 47 I define the four promises and the associated futures. In lines 51 - 54 each promise is moved to its own thread. A promise can only be moved but not copied. The threads execute the function `sumUp` (lines 18 - 25); `sumUp` takes a promise by rvalue reference as its first argument. In line 56, the futures ask for the result of the summation by using the blocking `get` call.