- Exercise

In this exercise, you will use std::thread::hardware_concurrency().

WE'LL COVER THE FOLLOWING

- Task 1

Task 1

Parametrize the summation of the natural number in the program below so that the number of threads depends on the value of `std::thread::hardware_concurrency()`.

> 💡 If you get the result 0, assume you have four threads.

```
// packagedTask.cpp

#include <utility>
#include <future>
#include <iostream>
#include <thread>
#include <deque>

class SumUp{
  public:
    int operator()(int beg, int end){
      long long int sum{0};
      for (int i= beg; i < end; ++i ) sum += i;
      return sum;
    }
};

int main(){

  std::cout << std::endl;

  SumUp sumUp1;
  SumUp sumUp2;
  SumUp sumUp3;
  SumUp sumUp4;
  // define the tasks
  std::packaged_task<int(int, int)> sumTask1(sumUp1);

  std::packaged_task<int(int, int)> sumTask2(sumUp2);
  std::packaged_task<int(int, int)> sumTask3(sumUp3);
  std::packaged_task<int(int, int)> sumTask4(sumUp4);

  // get the futures
  std::future<int> sumResult1= sumTask1.get_future();
  std::future<int> sumResult2= sumTask2.get_future();
  std::future<int> sumResult3= sumTask3.get_future();
  std::future<int> sumResult4= sumTask4.get_future();

  // push the tasks on the container
  std::deque< std::packaged_task<int(int, int)> > allTasks;
  allTasks.push_back(std::move(sumTask1));
  allTasks.push_back(std::move(sumTask2));
  allTasks.push_back(std::move(sumTask3));
  allTasks.push_back(std::move(sumTask4));

  int begin{1};
  int increment{2500};
  int end= begin + increment;

  // execute each task in a separate thread
  while ( ! allTasks.empty() ){
    std::packaged_task<int(int, int)> myTask= std::move(allTasks.front());
    allTasks.pop_front();
    std::thread sumThread(std::move(myTask), begin, end);
    begin= end;
    end += increment;
    sumThread.detach();
  }

  // get the results
  auto sum= sumResult1.get() + sumResult2.get() + sumResult3.get() + sumResult4.get();

  std::cout << "sum of 0 .. 10000 = " << sum << std::endl;

  std::cout << std::endl;

}
```

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