TP4 Calculating π using Riemann Sum in C++ with OpenMP

Michel Donnet

November 23, 2023

Contents

1 Explanation of the code

My code is all done in the main.cpp. We can execute my code like this:

```
./build/tp4
```

Or like this, with the number of circumscribed rectangles given

```
./build/tp4 1000000
```

Or like this, with the number of circumscribed rectangles and the number of processor given

```
./build/tp4 1000000 16
```

The default number of threads used is 8 and the default number of circumscribed rectangles is 10^8.

At the begining, I look after parameters, to set up the number of thread (stored in the variable 'n') and the number of circumscribed rectangles (stored in the variable 'nthread').

To change the default number of thread used, which is 8, I make:

```
omp_set_num_threads(nthread);
```

This function defined in 'omp.h' allow us to change the number of thread used to execute parallel sections of our code.

Then, I store the current time to compute at the end of the code the execution time like this:

```
// Begining of the code
double start = omp_get_wtime();
...
// End of the code
double end = omp_get_wtime();

// Print execution time:
cout << "Execution time: " << end - start << endl;
```

This function is also defined in 'omp.h' and give the current time in second, which is stored, in my case, in the variable 'start'.

I use a lambda expression to define the function $\frac{4}{1+x^2}$. Here I put a 4 instead of a 1, because the integration of the function $\frac{1}{1+x^2}$ in range [0, 1] give us an approximation of $\frac{\pi}{4}$, that's why I multiply the function $\frac{1}{1+x^2}$ by 4 to compute directly an approximation of π. My lambda function looks like this:

```
auto f = [](double x) -> double {
    return 4./(1. + x * x);
};
```

This code means that the lambda function f takes one parameter x and return a double, which is the computation of the function $\frac{4}{1+x^2}$ for the x given.

2 Explanation of the parallelization using OpenMP.

3 Presentation of the execution time results with varying thread counts.

4 Graphs depicting the scaling of execution times.

5 Observations and conclusions based on your findings.