# Notebook 2: `#pragma omp for`

Making the same thing, in an easier way

Gabriele Gaetano Fronzé

INFN Istituto Nazionale di Fisica Nucleare

UNIVERSITA' DEGLI STUDI DI TORINO

ELEMENTO Ready. Set. Cloud.

# Introduction

We would like to get rid of all the overhead on the for loops ranges we had to handle.

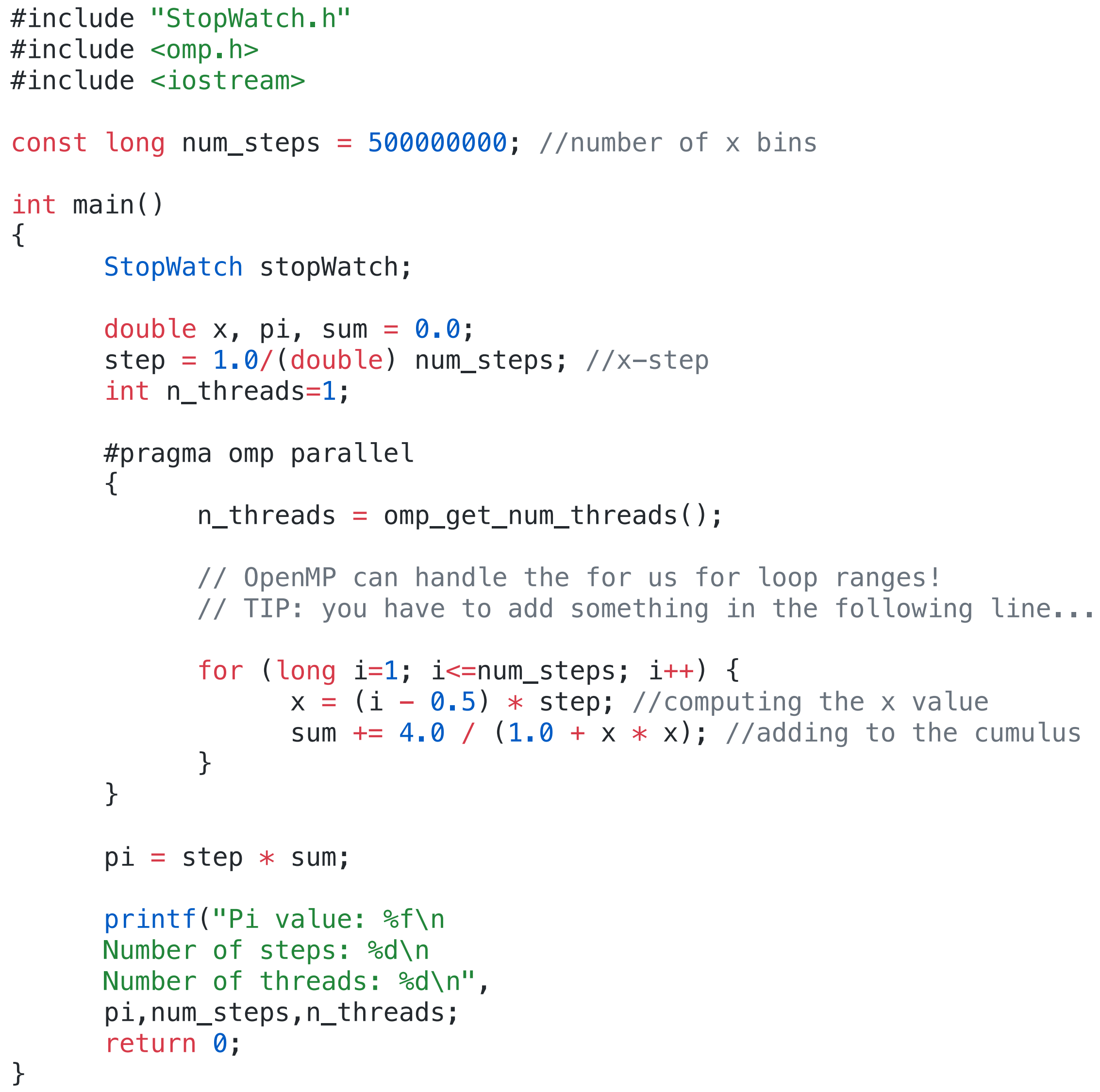
OpenMP allows the concatenation of constructs. The typical pragma:

```
#pragma omp parallel
```

Can be combined with:

```
#pragma omp for
```

To tell the compiler the following for-loop can be refactored by OpenMP, to execute the operations using the thread pool created by `parallel.`



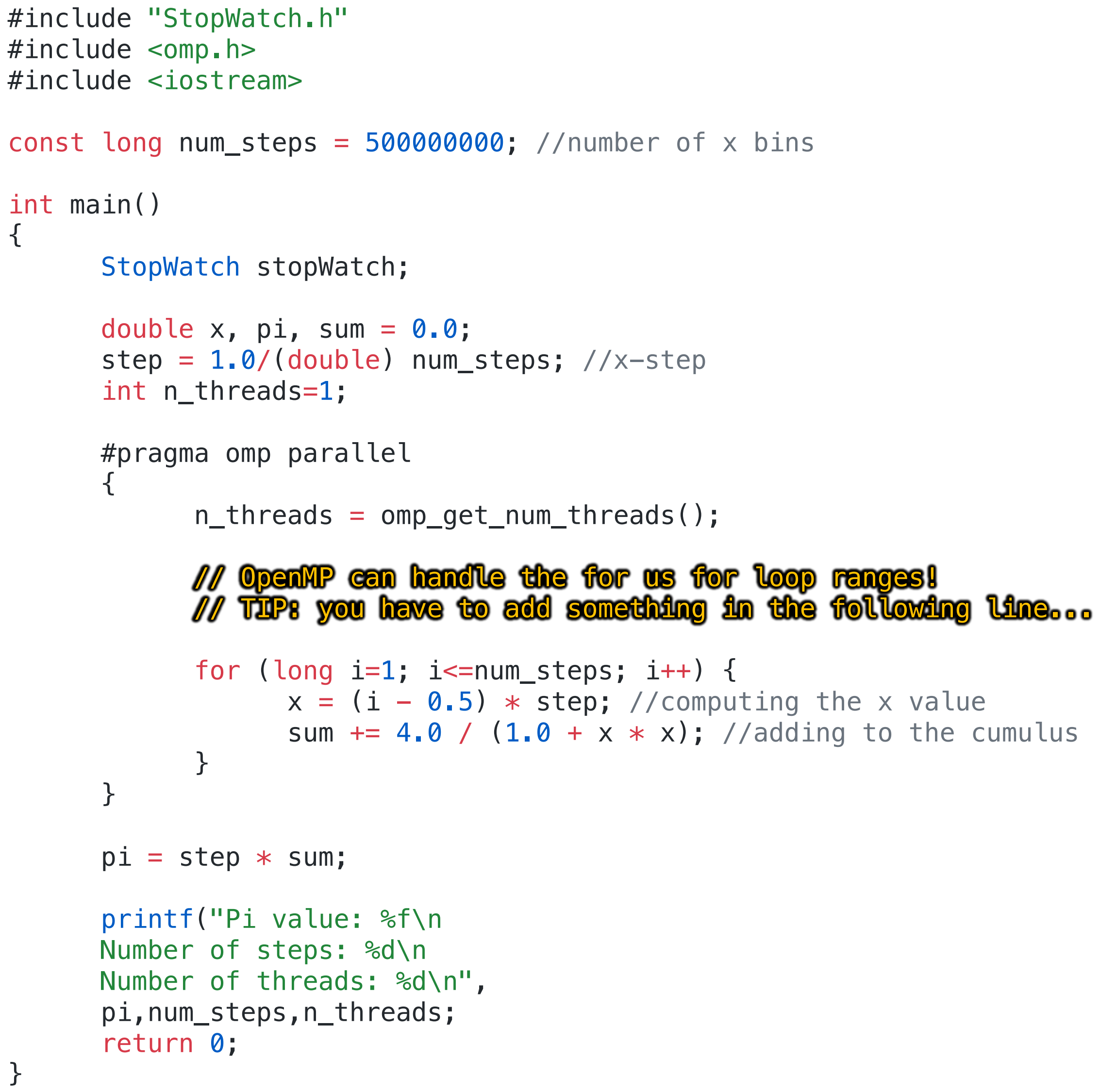
```
#include "StopWatch.h"
#include <omp.h>
#include <iostream>

const long num_steps = 500000000; //number of x bins

int main()
{
      StopWatch stopWatch;

      double x, pi, sum = 0.0;
      step = 1.0/(double) num_steps; //x-step
      int n_threads=1;

      #pragma omp parallel
      {
            n_threads = omp_get_num_threads();

            // OpenMP can handle the for us for loop ranges!
            // TIP: you have to add something in the following line...

            for (long i=1; i<=num_steps; i++) {
                  x = (i - 0.5) * step; //computing the x value
                  sum += 4.0 / (1.0 + x * x); //adding to the cumulus
            }
      }

      pi = step * sum;

      printf("Pi value: %f\n
      Number of steps: %d\n
      Number of threads: %d\n",
      pi,num_steps,n_threads;
      return 0;
}
```

# Introduction

Be aware that, if no code has to be put between the two pragmas, the directive can be written as:

```
#pragma omp parallel for
```

Since in this example and in the previous exercise we use `omp_get_num_threads()`, that returns 1 outside of parallel sections, we want to keep the two pragmas splitted.



```
#include "StopWatch.h"
#include <omp.h>
#include <iostream>

const long num_steps = 500000000; //number of x bins

int main()
{
    StopWatch stopWatch;

    double x, pi, sum = 0.0;
    step = 1.0/(double) num_steps; //x-step
    int n_threads=1;

    #pragma omp parallel
    {
        n_threads = omp_get_num_threads();

        // OpenMP can handle the for us for loop ranges!
        // TIP: you have to add something in the following line...

        for (long i=1; i<=num_steps; i++) {
            x = (i - 0.5) * step; //computing the x value
            sum += 4.0 / (1.0 + x * x); //adding to the cumulus
        }
    }

    pi = step * sum;

    printf("Pi value: %f\n
    Number of steps: %d\n
    Number of threads: %d\n",
    pi,num_steps,n_threads;
    return 0;
}
```

# And now again… Make the code rain!

Try to complete the exercise using `#pragma omp for`