Parallel Programming - Day 2

Nicola Meneghini

Exercise 1

The goal of the exercise is to approximate pi using the following equation:

$$\pi = 4 \int_0^1 \frac{1}{1+x^2} dx \simeq 4 \sum_{i=0}^{N-1} h \times f(x_{i+\frac{1}{2}})$$

In order to do that we need to define a function which we call `local_sum` and which calculates the sum on an interval $I \subset [0, 1]$. Its arguments will be:

- `local_a`: the beginning of the interval for a given thread
- `local_b`: the end of the interval
- `local_n`: the number of points at which I calculate the function $f(x_{i+\frac{1}{2}})$
- `h`: the total number of sub-intervals, or the height of each trapezoid.

The approximation was calculated using three approaches, which will now be explained. It is to be premised that in each case the result of the function was stored in the private variable `local_result`.

#pragma omp critical

This clause ensures that each thread have a mutually exclusive access to a block of code, thus avoiding race condition. In this way it is possible to safely sum `local_result` in `global_result`. Note that had we called the funciton inside clause there would have been no scaling.

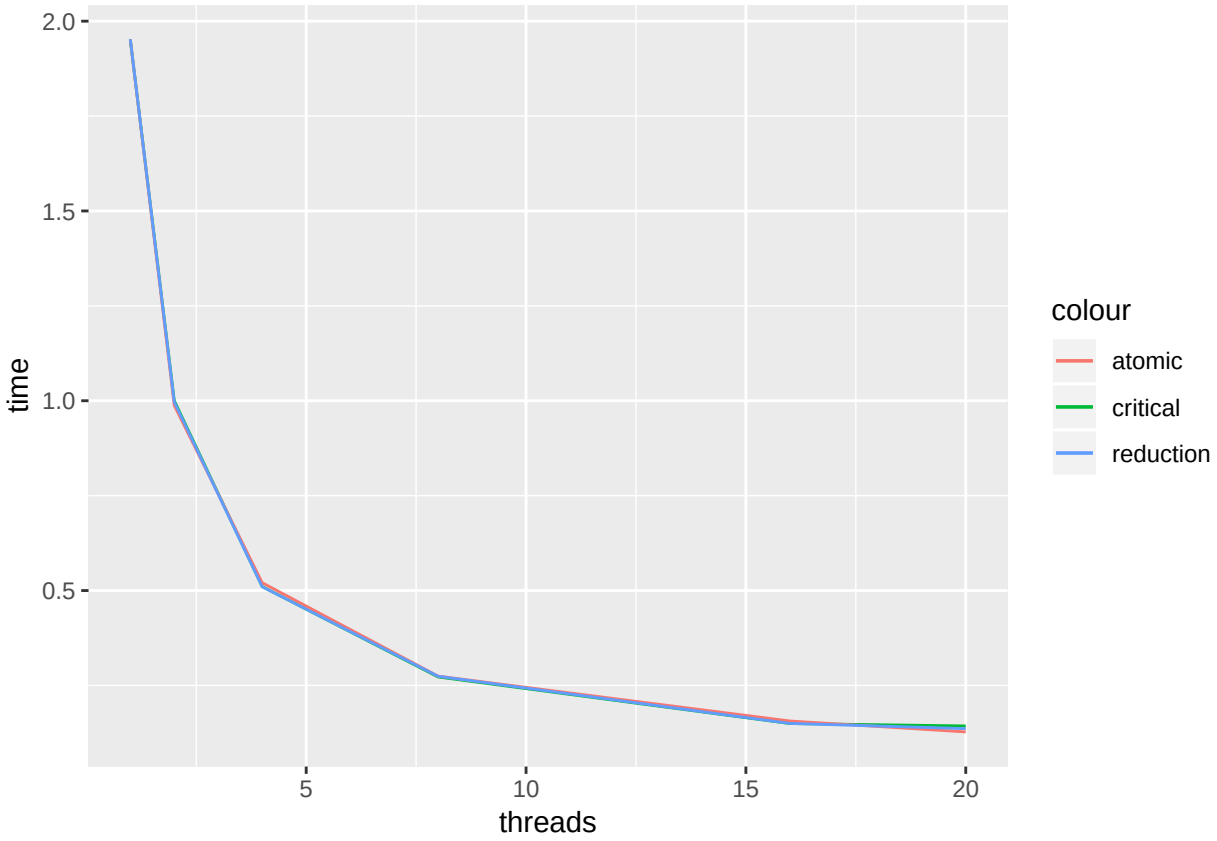
#pragma omp atomic

It has the same purposes of *critical*, but it is only allowed for simple operation, such as addition, multiplication and binary oerations. Since it only protects the read/update location, it is much faster than *critical*, but does not allow for more complex instruction like functions call.

#pragma omp parallel reduction(+:global_result)

This instruction creates a private copy of `global_result` for each thread, so that they will work on their own copy.

Below is reported a plot with the time taken to compute π in each case.



We can see that all the instruction lead to similar scaling results.

Exercise 2

We want to investigate the effects of using static schedules and dynamic scheduls dealing. We do that by using a special print function and different chunck size. The output of the trials can be found in the various files "loop_schedule.*"

Static Schedule

The call `schedule(static[,chunk])` divides block of iteration giving to each thread data of size chunck. This can be seen clearly in files "loop_schedule.static1" and "loop_schedule.static10" where we have groups of respectively 1 and 10 asteriscs. If the size is not specified, the load is evenly balanced automatically as we can see in "loop_schedule.static".

Dynamic Schedule

`schedule(dynamic[,chunk])` forces each thread to work on a chunk and immediatelly proceed to work on the following. Since defualt is chunk = 1, there should be no difference between the output in "loop_schedule.dynimic" and "loop_schedule.dynamic1".