HPC, Parallel Programming: Homework 1

Simone Amadio

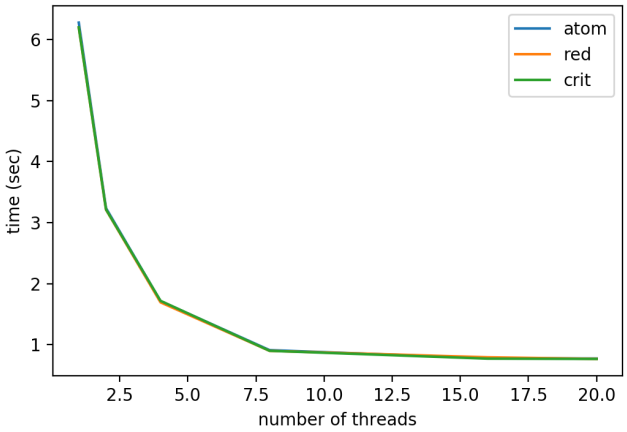
General informations

For clarity and brevity, the results will be presented mainly as graphs. The code and files containing the results are provided separately in this same folder. Only peculiar results will be analyzed singularly.

1 Exercise 1

1.1 N=10^9

In this exercise I tested the different behaviours of three OpenMP procedures to sum up a variable local to different threads: reduction, atomic and critical. In order to evaluate an instance of the problem I wrote the code for evaluating π using a series and submitted bash scripts to Ulysses.
I run bash scripts in order to test the different behaviour of each procedure using 1,2,4,8,16,20 threads and for each instance I did 5 tests to ensure result stability. In the following analysis only the median of the five times will be considered, noting that the single result for each number of threads presents little to no oscillation.
Using N=10^9 points for evaluating π, the following scaling curve is obtained

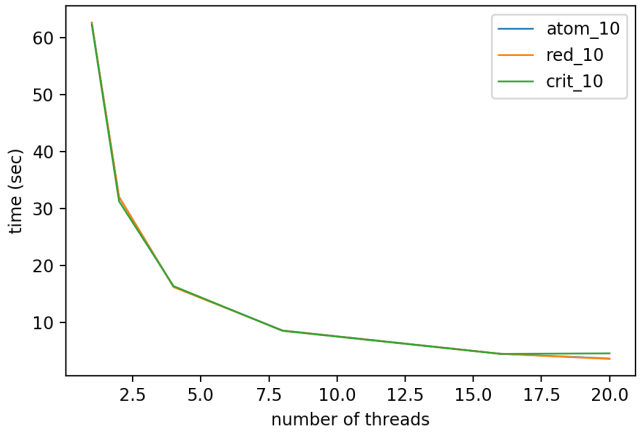


Scaling curve for N=10^9

As we can see, the three curves are overlapped: the three methods seem to be equivalent. Analyzing the numerical results as sum of all the tests, we get that reduction is the fastest method, atomic the slowest. Still, the difference is almost negligible.
Anyway, it's interesting to note that in a scaling curve we're actually interested to the speedup as rate. In this framework I considered as overall speedup the sum of all the individual rates $time(1)/time(n)$, with n= number of threads. The speedup is nearly the same for the three methods.

1.2 N=10^{10}

All of the results stated above are based on very similar results between the three methods. In order to see relevant difference, the same procedures have been applied changing N to 10^{10}. The results are the following:



Scaling curve for N=10^{10}

Here we can see a non negligible difference in the tail of the curve: reduction seems to be the fastest method, critical the slowest.
Analyzing the sum of times, we get the same results than before but with a non negligible difference. Using the speedup ratio, we get that reduction is the most effective, atom is almost equivalent to reduction, while the critical procedure has the worst scaling.

2 Exercise 2

In this exercise I used the provided function to see how, in a for loop, the different chunk sizes and schedules affects the division of the workload between threads. To do so, I parallelized the code between 10 threads, as required, and test the behaviour of each schedule using `chunk_size= None`,1 or 10.
The static schedule with `chunk_size=None` attempts a division of the workload as even as possible. This has been verified by using the provided `print_usage` function. A static schedule will always exploit all the spawned threads.

Specifying a chunk size in static schedule determines the length of each segment printed by the function `print_usage()`. At the end of the segment printed by the last thread, the whole process is repeated in a cyclic fashion.
The dynamic schedule instead generate more threads only in order to fill waiting times. Not specifying a chunk size makes the allocation of workload between the threads absolutely free.
Specifying a chunk size for dynamic schedule is equivalent to bind the next thread to execute at least `chunk size` operation before any other thread can operate. This sets the minimum length of the segment generated by the `print_usage` function.
An interesting output has been generated in dynamic schedule with `chunk size=None`: the function `print_usage` generated an almost straight line with respect to thread 1. This means that sometimes nearly all the workload can be handled by a single thread.
In this specific case we can suppose that the first thread working was the nearest to the allocated memory, so that almost no waiting time was encountered and only at the end another thread was involved.

Code and file references

- results of reduction for N=10^9 are stored in red3.txt, for N=10^{10} please confront red10.txt. Integer numbers represent the number of threads, double represents times. The results are obtained submitting the script red.sh. Same naming method applies to atom (atomic procedure) and crit (critical).
- Serial times are recorded in serial.txt
- Exercise 2 output is contained in static.txt or dynamic.txt files. The name corresponds to the schedule exploited. The chunk size is reported for each output.