Parallel computing - Exercise 1

Michela Venturini

Spring 2019

1 Compute `pi` by using OpenMP

The aim of the exercice is to approximate the value of pi using the midpoint formula. The code is implemented both in serial and then parallelized by using OpenMP.
The parallel code is implemented in three versions that exploit three different methods to avoid the *race conditions* that may be caused by a critical section of the code.

atomic The first implementation uses the `atomic` directive that prevents race conditions by enabling mutual exclusion for simple operation. It only protect read/update of target location. Where available, it takes advantage on the hardware providing an atomic increment operation so an atomic operation has much lower overhead with respect to the other solutions.

critical The second implementation uses the `critical` directive. It ensures that threads have mutually-exclusive access to a block of code and serialize the execution of the block. This directive is implemented in software and it is more expensive than the first solution but it is necessary if the target is to protect blocks of code involving more than one operation.

reduction The third implementation uses the `reduction` clause that creates a private variable for each of the threads and finally all threads' results are accumulated using operators.

2 Execution

The three implementations described are executed on Ulysses through a script (*ex1.sh*) for 1,4,8,16 and 20 threads and the time of execution is obtained by using `omp_get_wtime()`.
The execution is performed by submitting a job on Ulysses asking for a single node, since OpenMP is based on shared memory paradigm.

3 Results

The result of executions are stored in the file *results.txt*. The Figure 1 and Figure 2 show respectively the graphical representation of the timing for the different implementations by using `n=10e8` and the *Speedup* calculated as $S = \frac{T(1)}{T(N)}$. The performance of all implementations work similarly and scale well.

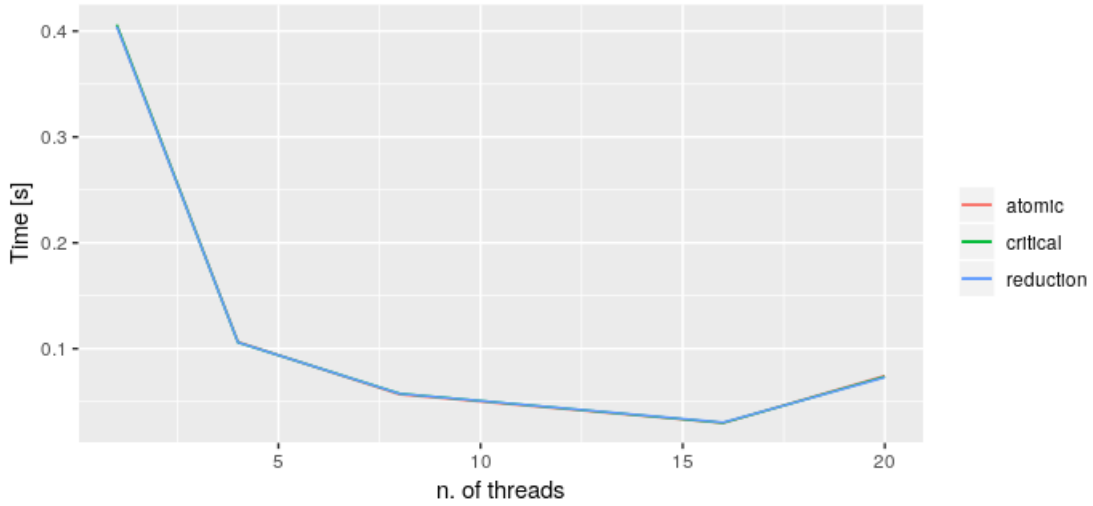

Figure 1: Comparison between parallel implementations

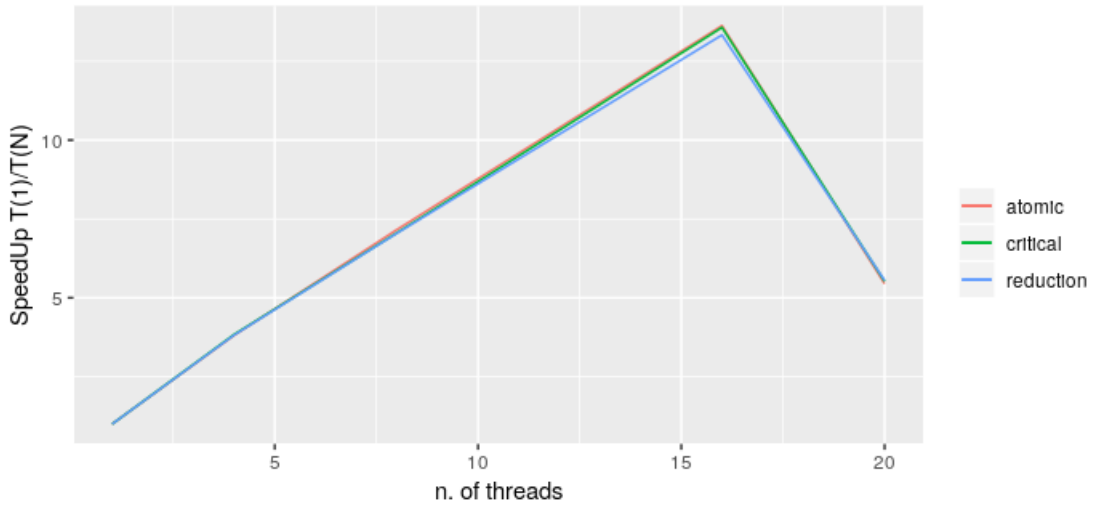

Figure 2: Comparison between parallel implementations(2)