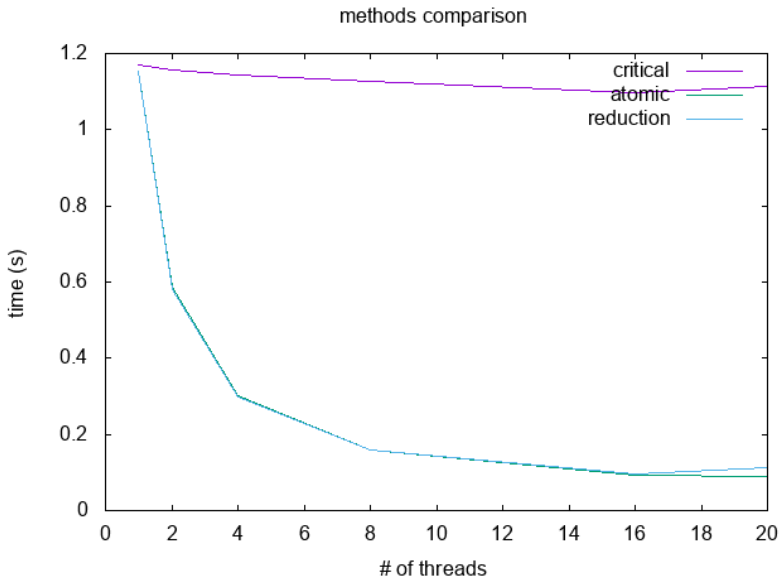
Giulia Franco
matricola SM3500370
Year 2018/2019
Exercise 1,Parallel Computing course.

Computing pi using openMP.

The aim of the exercise is to compute π using midpoints rule. The code is implemented using an openMP approach, exploiting the multithreads computation. In the code ($pi.c$) three different pragma sections:
-*critical*=A thread waits at the start of a critical region until no other thread in the program is executing a critical region with that same name.
-*atomic*=Allows access of a specific memory location atomically to threads. It ensures that race conditions are avoided through direct control of concurrent threads that might read or write to or from the particular memory location.
-*reduction*=Creates a private variable for each thread and finally all the resuts are accumulated using operations.
These methods are compared with respect to the serial computation using "$omp\_get\_wtime()$", using different number of threads.



The single timing for each method are represented in the current folder as *atomic.png*,*critical.png*,*reduction.png* with the corrisponding txt files.
From the upper graphic it's possible to notice the improvement of the computation using mutithreading from all the three sections. Specifically the critical secton is the one with less gain in performance since it's way more slower than the other ones.

loop schedule

The aim of the exercise is to create a visualization of two different OpenMP schedules using different chunks.
Aschedule in OpenMP is a specification of how iterations of associated loops are divided into contiguous non-empty subsets (*chunk*) and are distributed to threads. There are different kind of chunks, we'll study:
static=iterations blocks are mapped statically to the execution threads. OpenMP run-time guarantees that each thread will receive exactly the same iteration range .
dynamic=threads may receive different iteration range.
The output of the exercise is the the file "$output\_loop_schedule\_T\_10$" using 10 threads and "$output\_loop_schedule\_T\_4$" using 4 threads.

Compiling and Executing exercises

-Exercise 1
The first step on Ulysses is to reserve a node for the execution:
qsub -l nodes=1:ppn=20 -I -l walltime=1:00:00.

Then compiling using:
module load intel
icc -qopenmp pi.c -o pi Finally executing using the script in the current

folder:
./cases.sh

-Exercise 2
compiling using:
module load intel
icc -qopenmp *loop_schedule.c* -o *loop_schedule*.

Executing:
./loop_schedule