**LAB2 SESSION 1: A very practical introduction to OpenMP (I)**

**1.1 OpenMP questionnaire (Day 1: Parallel regions and implicit tasks)**

* 1.hello.c

1. How many times will you see the "Hello world!" message if the program is executed with "./1.hello"?
2. Without changing the program, how to make it to print 4 times the "Hello World!" message?

* 2.hello.c:

1. Is the execution of the program correct? (i.e., prints a sequence of "(Thid) Hello (Thid) world!" being Thid the thread identifier). If not, add a data sharing clause to make it correct?
2. Are the lines always printed in the same order? Why the messages sometimes appear inter-mixed? (Execute several times in order to see this).

* 3.how many.c: Assuming the OMP NUM THREADS variable is set to 8 with "OMP NUM THREADS=8 ./3.how many"

1. What does omp get num threads return when invoked outside and inside a parallel region?
2. Indicate the two alternatives to supersede the number of threads that is specified by the OMP NUM THREADS environment variable.
3. Which is the lifespan for each way of defining the number of threads to be used?

* 4.data sharing.c

1. Which is the value of variable x after the execution of each parallel region with different data-sharing attribute (shared, private, firstprivate and reduction)? Is that the value you would expect? (Execute several times if necessary)

* 5.datarace.c

1. Should this program always return a correct result? Reason either your positive or negative answer.
2. Propose two alternative solutions to make it correct, without changing the structure of the code (just add directives or clauses). Explain why they make the execution correct.
3. Write an alternative distribution of iterations to implicit tasks (threads) so that each of them executes only one block of consecutive iterations (i.e. N divided by the number of threads).

* 6.datarace.c

1. Should this program always return a correct result? Reason either your positive or negative answer.
2. Propose two alternative solutions to make it correct, without changing the structure of the program (just using directives or clauses) and never making use of critical. Explain why they make the execution correct.

**1.2 Observing overheads (no acabat)**

Take a look at the four different versions and make sure you understand them. How many synchronisation operations (critical or atomic) are executed in each version?

pi\_omp\_critical: Total overhead when executed with 100000000 iterations on 1 threads: 3774829.0000 microseconds.

1. If executed with only 1 thread and 100.000.000 iterations, do you notice any major overhead in the execution time caused by the use of the different synchronisation mechanisms? You can compare with the baseline execution time of the sequential version in pi sequential.c.
2. If executed with 4 and 8 threads and the same number of iterations, do the 4 programs benefit from the use of several processors in the same way? Can you guess the reason for this behaviour?