**LABORATORY 2: A very practical introduction to OpenMP**

**1.1 OpenMP questionnaire**

* 1.hello.c

1. How many times will you see the "Hello world!" message if the program is executed with "./1.hello"?

We see it 2 times, because there are two threads executing the same code.

1. Without changing the program, how to make it print 4 times the "Hello World!" message?

OMP\_NUM\_THREADS=4 ./1.hello

* 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?

We have to add a critical clause inside the loop but outside the if condition.

1. Are the lines always printed in the same order? Why do the messages sometimes appear inter-mixed? (Execute several times in order to see this).

It’s not always in the same order, because the id variable is not private.

* 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?

The number of created threads in the parallel section when invoked outside a parallel section is always 1 and inside it’s the number of active threads.

1. Indicate the two alternatives to supersede the number of threads that is specified by the OMP NUM THREADS environment variable.

1- #pragma omp parallel num\_threads()

2- omp\_set\_num\_threads()

1. Which is the lifespan for each way of defining the number of threads to be used?

1- Lasts until the next #pragma omp parallel

2- Lasts while the loop is not over

* 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)

* Shared: can be any number between 0-31
* Private and firstprivate: can be only 5 because the result doesn’t “go outside”, it can’t have its value changed outside the pragma omp (first)private clause.
* Reduction: can only be 501, because it’s the sum of the partial sums.
* 5.datarace.c

1. Should this program always return a correct result? Reason either your positive or negative answer.

No, because the maxvalue variable suffers dataraces.

1. 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.

* **reduction (max:maxvalue):** it works because the maximum is a commutative/associative operation, so the partial result can be combined in a complete one.
* The code will be slower, but we can also solve the problem by writing #pragma omp critical inside the for loop and outside the if condition.

1. 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).

if (i=id\*n/howmany; i<(id+1)\*n/howmany; i++)

* 6.datarace.c

1. Should this program always return a correct result? Reason either your positive or negative answer.

No, there are dataraces in countmax.

1. 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.

* #pragma omp parallel private(i) reduction(+:countmax)
* Inside the if condition and before the sentence countmax++ we need to write #pragma omp atomic

**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.c**: Executes 4 times the critical operation.

**pi\_omp\_atomic.c**: Executes 4 times the atomic operation.

**pi\_omp\_sumlocal.c**: Executes 4 times the critical operation. Although the section is smaller than in the first one.

**pi\_omp\_reduction.c**: Does not execute any critical/atomic operation

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.

* critical: Total overhead when executed with 100000000 iterations on 1 threads: 1862184.0000 microseconds
* atomic: Total overhead when executed with 100000000 iterations on 1 threads: -4178.0000 microseconds
* sum\_local: Total overhead when executed with 100000000 iterations on 1 threads: -3949.0000 microseconds
* reduction: Total overhead when executed with 100000000 iterations on 1 threads: -1916.0000 microseconds

1. 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?

* critical:
  + Total overhead when executed with 100000000 iterations on 4 threads: 50401203.7500 microseconds
  + Total overhead when executed with 100000000 iterations on 8 threads: 43830557.8750 microseconds
* atomic:
  + Total overhead when executed with 100000000 iterations on 4 threads: 7649572.5000 microseconds
  + Total overhead when executed with 100000000 iterations on 8 threads: 9683089.5000 microseconds
* sum\_local:
  + Total overhead when executed with 100000000 iterations on 4 threads: 10273.0000 microseconds
  + Total overhead when executed with 100000000 iterations on 8 threads: 16040.8750 microseconds
* reduction:
  + Total overhead when executed with 100000000 iterations on 4 threads: 10932.5000 microseconds
  + Total overhead when executed with 100000000 iterations on 8 threads: 15398.6250 microseconds

Conclusions:

Overall, the executions using one thread have been the ones