PROGRAMACIÓN PARALELA
Intermediate OpenMP Tutorial

In the accompanying archive, there is a compressed directory called `Lab5`. Once the archive is uncompressed change to that directory. Inside it there is a `Makefile`. Use `make` to compile the programs in that directory.

The first program you should execute is called `for3`. It simply illustrates how to print out all pairs of integers from `1` to `max` in no particular order, something that calls for two nested `for` loops. As all iterations of both nested loops are independent, either loop can be parallelized while the other is not. This is achieved by placing the `omp parallel for` directive in front of the loop targeted for parallelisation. In this case, we will put the directive associated to the outer loop. We already know that the compiler will generate code to partition the job between the generated threads. The question is how balanced the workload is?

The next program to test is called `for4`. In this case, the `omp parallel for` directive is applied to both loops.

The next program to test is called `for5`. In this case, the `omp parallel for` directive is applied to the inner loop.

Finally, in `for6`, we apply the `collapse` clause, which has a parameter (`2` in this case) that indicates how many loops to combine.

What can we conclude from this?

Data parallelism is typically found in loops, and hence, OpenMP tries to exploit it. Nevertheless, no matter how good is the generated code, by using OpenMP we are not able to scale up beyond the frontiers of the node. And, we know that it might be quite constrained for heavy workloads.

Therefore, in order to tackle bigger workloads, we need to combine scaling out (i.e., using MPI) and scaling up (i.e., using e.g., OpenMP). This approach is illustrated in `mpi_omp0`.

To compile a MPI + OpenMP program you will need to use the `mpicc` compiler and pass to it the `-fopenmp` option flag, which enables code generation using the OpenMP API.

The last program to try, `cpit0`, shows how to combine MPI collective operations and OpenMP directives to estimate the value of `pi` by means of the integral of `4 / ( 1 + x^2 )` using tangent-trapezoidal rule using `n` trapezoids.

What do you think of the approach used to measure time? Can you figure it out a better approach?