Lab 2. OpenMP – operations with matrices and performance studies

Main goals of the assignment

- Learn about the compiler directives for multi-core system for computational-intensive tasks;
- Understand the differences between global and thread-private variables;
- Learn about the problem dimensions that are requiring parallel computations;
- Learn how to do a performance study for a parallel algorithm implementation.

The problem to solve

General overview

A parallel version of matrix to matrix multiplication that is efficient is requested.

Background

Let A and B be to $n \times n$ matrices of real numbers. The elements of the product of the two matrices, noted by C in what follows is computed by

$$c_{ij} = \sum_{k=0}^{n-1} a_{ik} b_{kj}, \text{ for } 0 \leq i, j < n \tag{1}$$

The time complexity of this multiplication is $O(n^3)$. When n is large, then the computational time is very high.

How to parallelize

See the slide for Lecture 4 for some examples. We will take the case in which we have p processing elements that are available (e.g. the number of our desktop cores).
In this exercise, we will split the matrix A in horizontal slices and we will assign one slice or more to one core. The core m $(0 \leq m < p)$ will therefore deal with the rows $m \cdot \frac{n}{p} \leq i < (m+1) \cdot \frac{n}{p}$ of A, but also of C. Note that is no overlaps of readings/writing from/to the global memories as was the case in Lab 1). (Be carefull to the use of loop variables!)

To do

1. Write the sequential code to multiply the two matrices. The dimension of the matrices should be a given as parameter in the command line (hint: argv[1]). Dynamic allocation should be used (hint: see how we allocate the space for the vectors in lab 1). The two matrices' elements should be set to values that allows the simple check of the results (hint: $a_{ij} = b_{ij} = 1$).
2. Write the parallel code to multiply the two matrices (hints: use the pragma for the external loop; be carefull how the loop variables are shared/or not). The number of cores should be a parameter in the command line (hint: argv[2]). Check the correctness of the result in simple cases.
3. Introduce time records (hint: omp_get_wtime) before and after the part that is parallelized.
4. Record the times $T_p^{(n)}$ in table like the folowing (with the maximum cores that you have, e.g. 8 or 16) and compute the speedup
5. Compute the speedups using

$$S_p^{(n)} = \frac{T_1^{(n)}}{T_p^{(n)}}$$

   and record them in a similar table with the above one. Due the same for the efficiency $E_p^{(n)} = S_p^{(n)}/p$.

Table 1: Put inside the boxes the recorded times

$n\backslash p$	1	2	4	8
1600				
2000				
2400				

6. Display in a graphic the values of S as dependence on p (respectively E in another graphic) and with different polygonal lines (and colors) the values for different n

7. Draw conclusions related to:

   - increase/decrease of S with p;
   - dependence of the problem dimension n