PDP Lab 3 - Rad George-Rares 936/2

Problem:

Divide a simple task between threads. The task can easily be divided in sub-tasks requiring no cooperation at all. See the caching effects, and the costs of creating threads and of switching between threads

Write several programs to compute the product of two matrices.

Have a function that computes a single element of the resulting matrix.

Have a second function whose each call will constitute a parallel task (that is, this function will be called on several threads in parallel). This function will call the above one several times consecutively to compute several elements of the resulting matrix. Consider the following ways of splitting the work between tasks (for the examples, consider the final matrix being 9x9 and the work split into 4 tasks):

1. Each task computes consecutive elements, going row after row.
2. Each task computes consecutive elements, going column after column.
3. Each task takes every *k*-th element (where *k* is the number of tasks), going row by row.

For running the tasks, also implement 2 approaches:

1. Create an actual thread for each task (use the low-level thread mechanism from the programming language);
2. Use a thread pool.

Tests:

Simple Thread Implementation:

Data Amount	No. threads	Row Computation	Col Computation	K-th Computation
2x2	2	0.040s	0.045s	0.042s
4x4	4	0.046s	0.049s	0.051s
9x9	4	0.07s	0.10s	0.084s

Pool Thread Implementation:

Data Amount	No. threads	Fixed thr number	Row Computation
9x9	4	4	0.017s
9x9	4	2	0.019s
9x9	4	1	0.02s

As we can see from the second table, the thread pool computations depend a lot on how you spread the tasks, and what is the fixed number of worker threads.