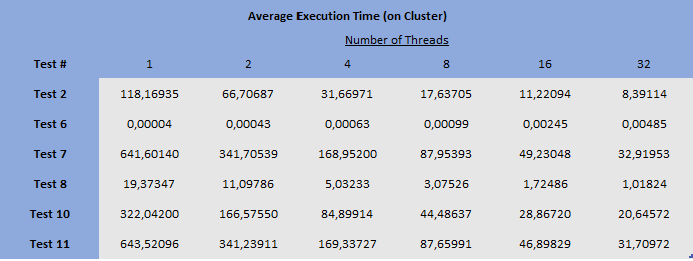
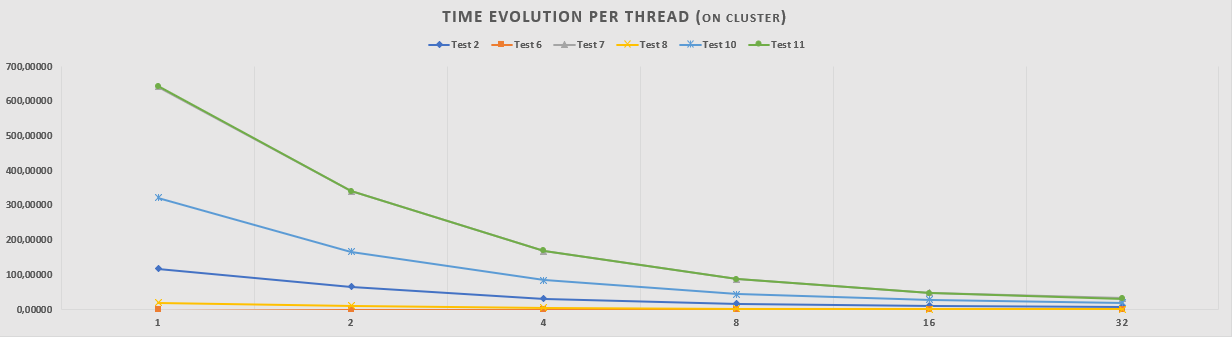
**Analysis**

**Cluster (Tests 2, 6, 7, 8, 10, 11)**

The tests 2, 6, 7, 8, 10 and 11 were run on the cluster with 1, 2, 4, 8, 16 and 32 threads. The resulting average execution times and their evolution per number of threads are displayed in the table and graph below:

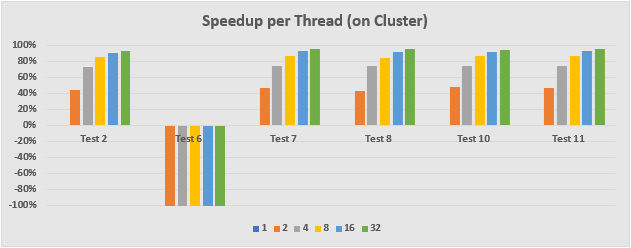




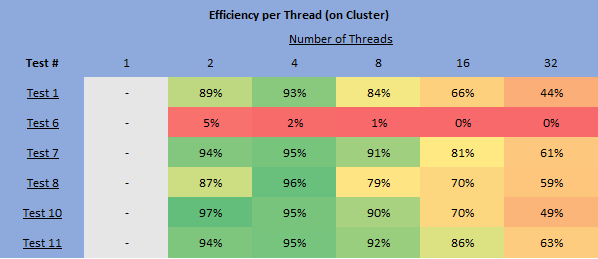
(O TESTE 7 ESTÁ ESCONDIDO ATRÁS DO 11 -> É PROBLEMA OU PERCEBE-SE PELA TABELA?)

All tests see their execution times reduced significantly as the number of threads increases except for test 6, because its sequential version already had a very small execution time and the performance gains could not outweigh the parallelization costs, such as the opening and closing of threads.

Comparing the registered values per number of threads to the sequential version of each test, the Speedup was determined. As illustrated below, all tests (except test 6) achieved great improvements, especially when increasing the number of threads from 1 to 2 (40-50%) and from 2 to 4 (20-30%). After that, as the number of threads increases, the Speedup gains become progressively less pronounced.



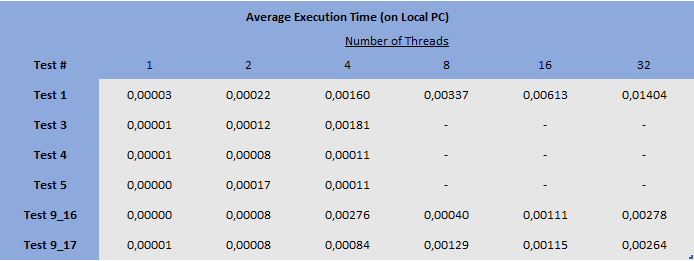
Dividing the aforementioned Speedup by the number of processors in which the program was run, the Efficiency per number of threads was established. Its values are displayed in the table below:



Other than test 6, whose bad results were already expected (as mentioned in the “Testing and Results” section), it’s clear that great efficiency values are achieved with 2 and 4 threads, the latter’s being the best values of the whole batch, and that increasing the number of threads further than 8 is quite inefficient as the speedup gains do not measure up.

**Local (Tests 1, 3, 4, 5, 9\_16, 9\_17)**

The tests 1, 3, 4, 5, 9\_16 and 9\_17 were run locally and with 1, 2, 4, 8, 16 and 32 threads as well. The resulting average execution times per number of threads are displayed in the table and graph below:



As registered in the cluster for test 6, all the tests chosen to be run locally see their execution times worsen as the number of threads increases because this tests’ purpose is not to evaluate the performance of the program but debugging and testing its correctness. Since the data to be computed is relatively small the parallelization costs outweigh the parallelization gains.