NUMERICAL SCIENTIFIC COMPUTING: Mini project 1

Lars Depuydt - CE8-NDS

The assignment

The miniproject is about computing the Mandelbrot set. This can be done in several ways, one faster than the other. Computing this requires a loop to run over each value of a complex matrix, which makes the problem O(x^3).
For each point in the matrix, we must check whether it is stable using the Mandelbrot quadratic complex mapping. This is done for a number of iterations I and with a threshold T. The set is calculated within a predefined area and density.

Runtime

Multiple implementations of the problem were made and their runtime is compared against each other in the following table. All the implementations were tested with the following parameters:

x_min, x_max, y_min, y_max = -2.0, 1.0, -1.5, 1.5
p_im, p_re = 5000, 5000
I = 30
T = 2

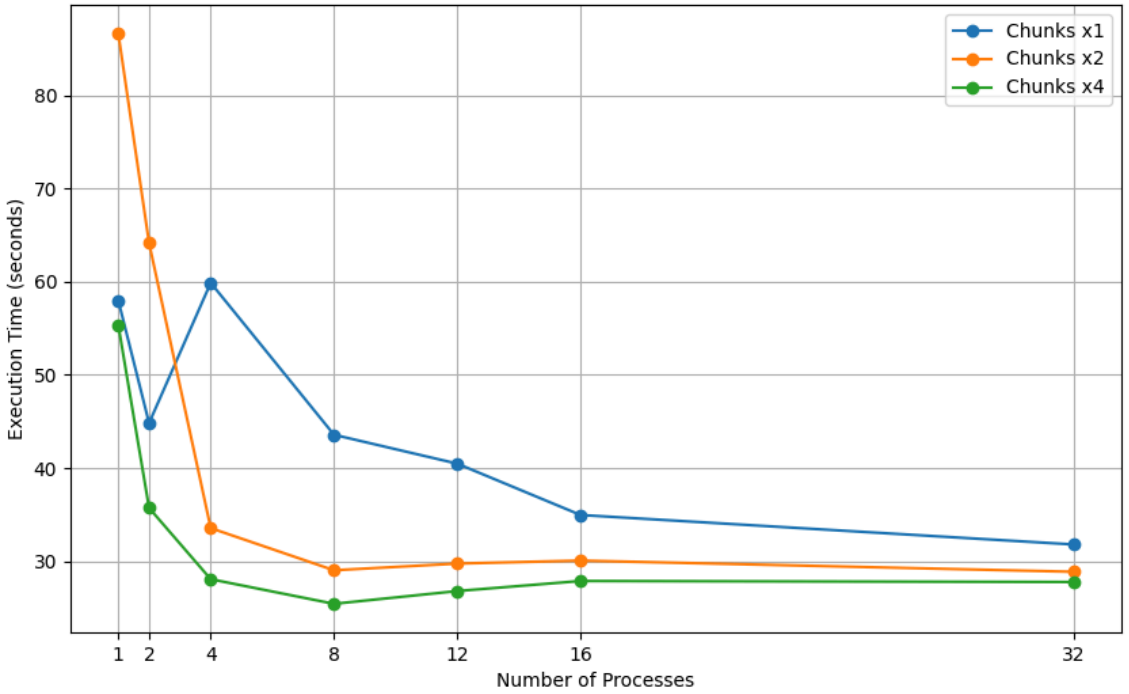
	Run 1 (s)	**Run 2 (s)**	**Run 3 (s)**	**Average (s)**
Naive (loops)	79,04	90,20	82,81	**84,02**
Numba-optimized version(s)	5,19	5,75	5,51	**5,48**
Numpy (vectorized)	17,66	18,03	17,08	**17,59**
Parallel version using multiprocessing (8 processes, 48 chunks)	24,61	29,50	29,76	**27,96**

It can be seen in the table above that the naive version is way slower than all three optimized versions.

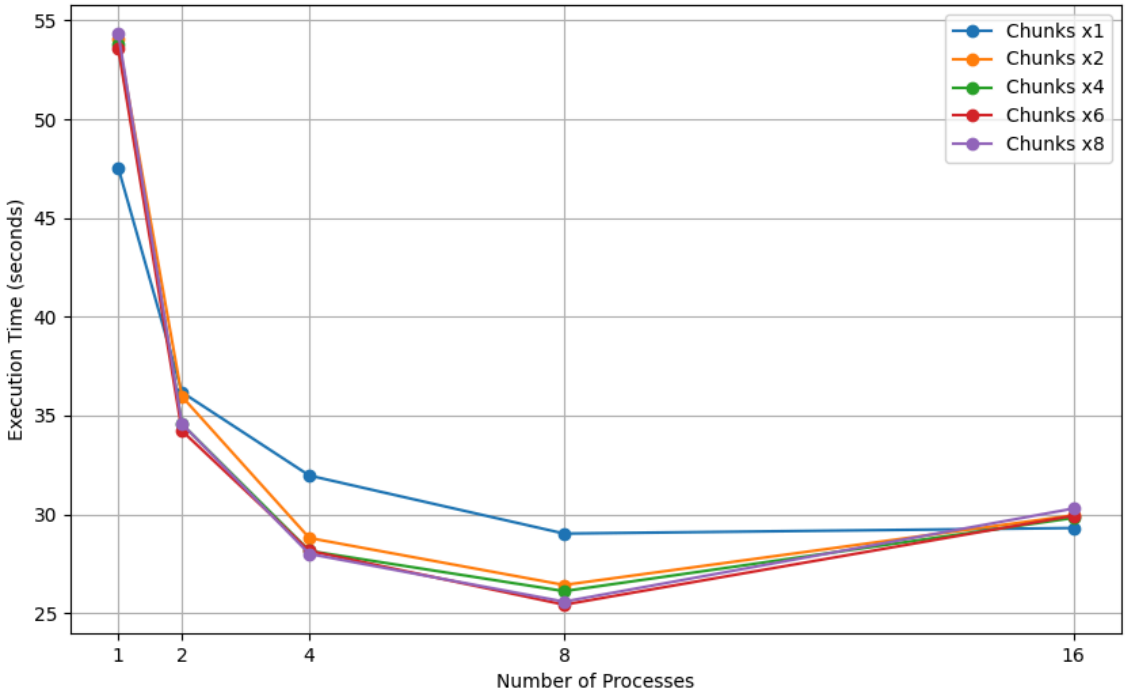
- If the calculation is parallelized using 8 cores, then we gain a speedup of almost x3. This is significant, but it is important to note that this is still far from a x8 speedup, as one could think when using 8 cores instead of one. This has to do with overhead, not everything is parallelized…
- Next, the Numpy vectorized version gains a speedup of x5, which is quite significant. This has to do with NumPys optimizations for working with (big) matrices.
- Finally, the Numba-optimized code performs the best. This with a speedup of almost x17. This has mostly to do with C code being way for performant than Python code, used by the just-in-time compiler of Numba.

Multiprocessing analysis

When multiprocessing, the number of processes and chunks can be altered to increase the optimal speed. This is mainly computer-specific. The following tests were run on a computer with 8 cores.



Here the number of processes and chunks were varied and their influence on the execution time was measured. The chunk size depends on the number of processes used in that specific test (1x, 2x… the number of processes).
It can be seen that how closer the number of processes goes in the direction of 8, the faster the results. Moreover ‘x4’ chunks seem to perform the best. Another test is performed to better understand these results.



The same test is performed again, but now also for ‘x6’ and ‘x8’ chunk sizes. It can be seen that 8 processes still perform the best and that the ‘x6’ chunk size seems to yield the best result, although the results lie close to each other. This is why we chose this result as the one used in the runtime comparison table.