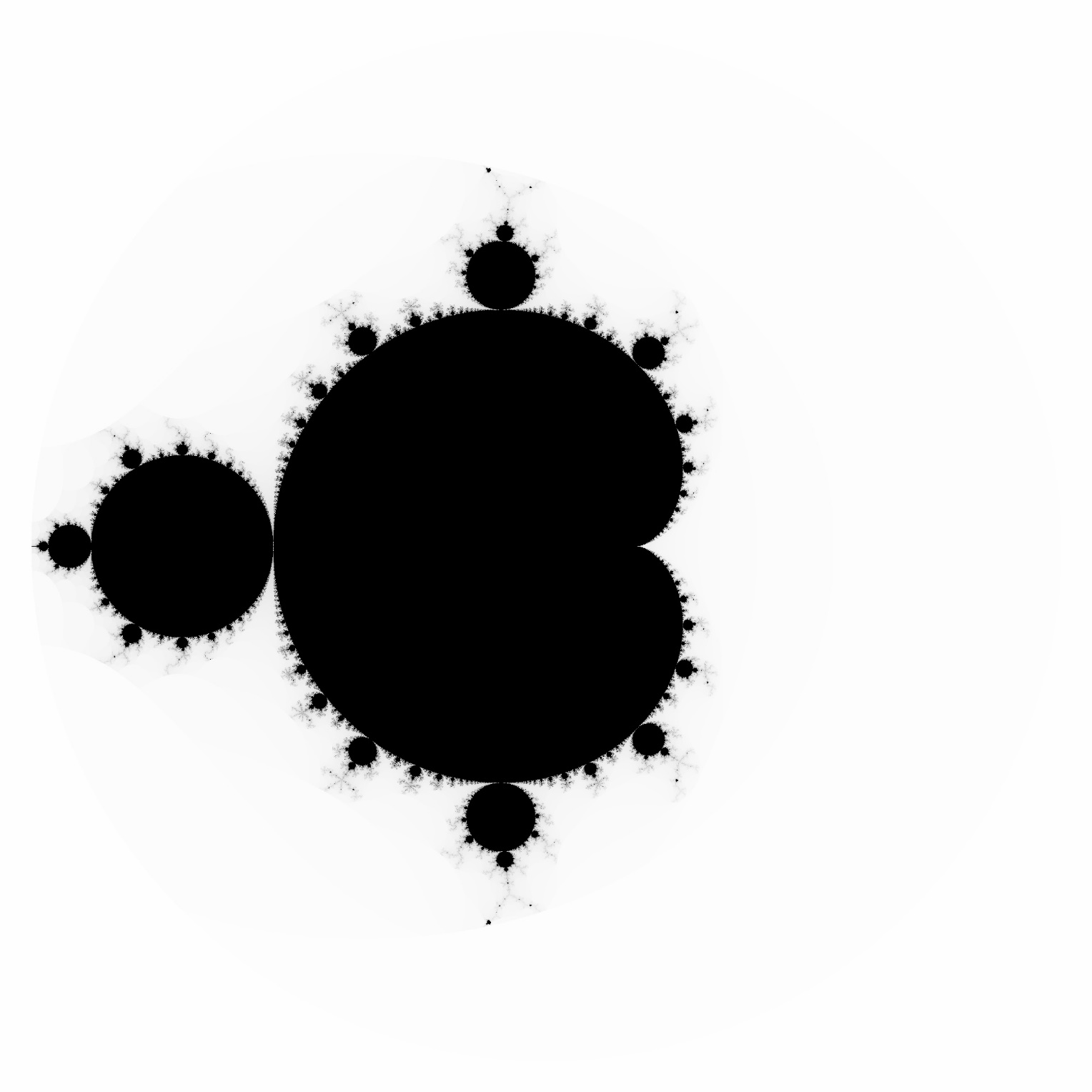
**Third Assignment Report**

**Exercise 1**

**Computation of the Mandelbrot set**

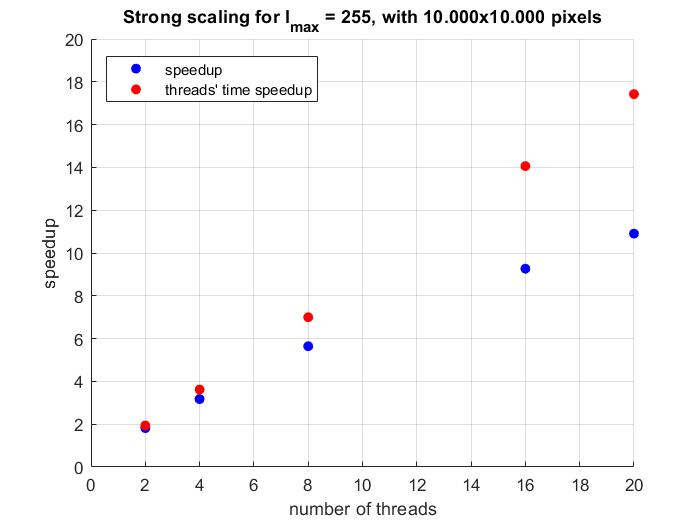
The assignment was to compute the Mandelbrot set, represented in the figure below.



All tests were conducted by considering the square region with vertices (-1.5, -1.5),

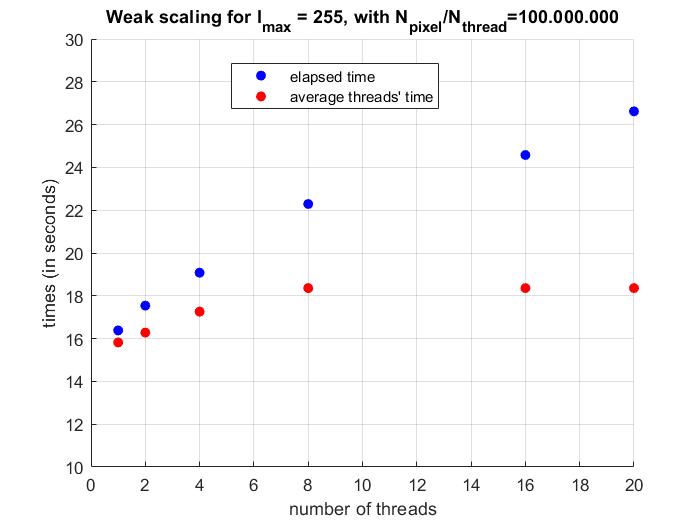
(-1.5, 1.5), (1.5, 1.5), (1.5, -1.5), in order to compute the whole Mandelbrot set. In order to achieve a balanced workload a dynamic scheduling was necessary for the loop. Moreover, on Ulysses, the standard setting is OMP\_PROC\_BIND=FALSE and OMP\_PLACES=ʻʼ, which in some cases resulted in the system migrating threads, therefore causing degraded performances. Hence it was necessary to set OMP\_PLACES=cores to avoid such problems.

In the picture below we can see a strong scaling test, done with a problem size of 10.000x10.000 pixels and a maximum of 255 iterations per pixels.



Blue dots represent the speedup calculated on the elapsed times, whereas the red ones were done considering only the average walltime of the threads. As a result the whole algorithm doesn’t scale too well, probably due to the fact that the creation of the image is not done in parallel. On the other hand the threads time, being the problem embarrassingly parallel, should have linear speedup. This doesn’t exactly happen, even though the speedup is still in the worst case 85% of the theoretical one, due to the parallel overhead caused by the dynamic scheduling of the OpenMP for loop. This is needed in order to ensure a balanced load distribution among the threads, as with a static scheduling the results were horrendous, with some threads taking up to 40 seconds and other taking 0.002 seconds. Guided scheduling, albeit better than the static one, still delivered worse performances than dynamic scheduling.

In the figure below we can see a weak scaling test, done by fixing a problem size of 100.000.000 pixels per thread and a maximum of 255 iterations per pixels.



For the algorithm to weakly scale well the execution times should be more or less the same, which clearly is not the case with this application. The threads’ times scale decently, whereas the elapsed time don’t really scale. This is definitely due to the fact that by increasing the problem size also the serial part increases, and in particular writing to the file requires more time. Of course by increasing the number of pixels also the overhead increases, as the OpenMP scheduler has more work to do, and that contributes as well to the bad scaling of the code.

The next tests were done with 65535 as the maximum number of iterations. The algorithm in general scaled better than before, definitely due to the computation being the main part of the execution.

