Your report (PDF) and the required sources (ZIP) must be handed in electronically!

Deadline: latest on Sunday, October 19, 2025 at midnight!

Project 1 – Mandelbrot

In our first project of Large Scale Modelling we will implement the Mandelbrot set calculation.

The Mandelbrot set can be calculated for a given pixel using a truncation function:

$$p_0 = x + iy \tag{1}$$

$$p_{n+1} = p_n^2 + p_0 \tag{2}$$

where x and y are the image coordinates, and i is the imaginary number. The iteration is truncated when $|p_n| > 2$ or $n > n_{\max}$. Then store n (or use some color palette of the number of iterations) in the image.

The outer-most Mandelbrot set is defined in the image limits:

$$x \in [-2.2, 0.75] \tag{3}$$

$$y \in [-1.3, 1.3]. \tag{4}$$

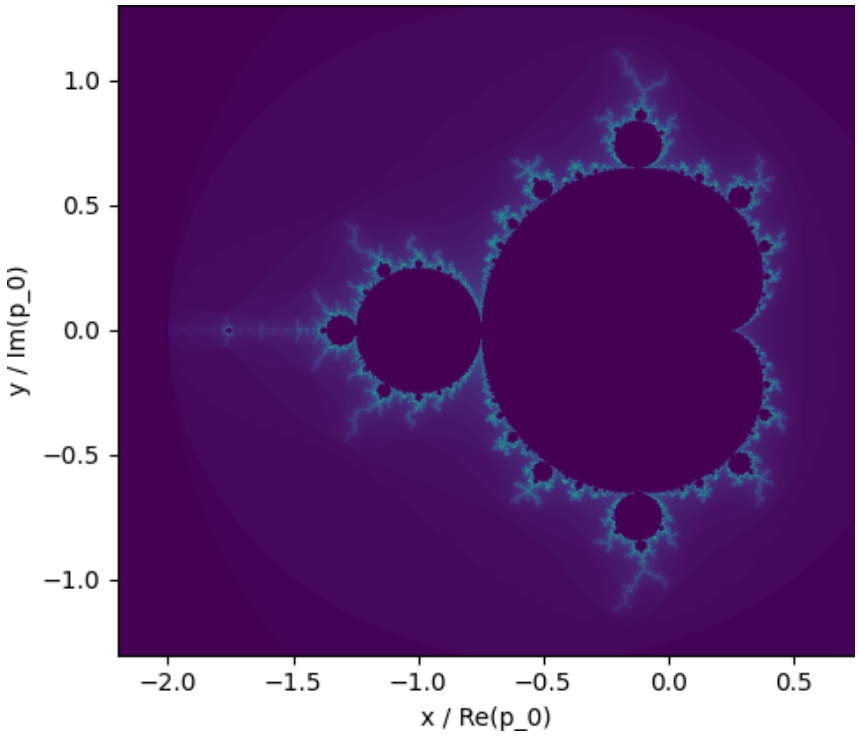

Figure 1: An image of the Mandelbrot set in the default view-range.

Please extend the shipped `Mandelbrot.py` code to use MPI and if additional arguments are necessary, add them to the end of the argument list.

With the current flags you can investigate several different regions of the image and thus scour its inherent load imbalance problems.

Expected investigations

The Mandelbrot example is a case of a compute-bound problem with load-imbalance. We expect you to at least have 2 different MPI implementations:

- a blocking send/receive code
- a non-blocking send/receive code

In both cases they should be implemented using a row-only distribution, i.e. the image is updated along the rows (x coordinate) of the array. You may later investigate/describe other work distributions. Use the `chunk-size` variable in the shipped code to create an initial load-distribution algorithm. We expect that you present a report with at least the following surveys:

- Discuss, implement and analyse different load balancing strategies.

  For instance, one could implement a static scheduler (each rank gets a pre-determined chunk-segment), or one could implement a dynamic scheduler (where work is *requested* by the workers).

  There is a lot of freedom in choosing how to distribute work, *remember* to describe the distribution in common language.

  > We prefer deep analysis of a few distributions vs. a weak analysis on many distributions! This is not a programming competition!

  If there is not enough time for implementing a dynamic scheduling, discuss how one could imagine it being implemented.
- A performance benchmark of the different implementations. E.g. scalability plots, possibly with different load balancing strategies.

  > Explain why you get the plots you get!
- A performance benchmark when using 1 node vs. multiple nodes. Hint: the size of the image, i.e. the number of pixels, has an influence on different performance aspects here!
- Discuss code-progress and changes in the code by showing small code-snippets in the report, and how the changes influence the performance (either worsen or improves).

Notes

- Labels on figures, and explanatory figure captions!
- Ensure your lines on graphs are distinguishable.
- Consider what you are timing, and what you compare.
- Please write the report so your fellow students (not taking the course) can understand it.
- Ensure to explain why you see things, if a scalability plot that flattens has a reason, what's the reason? Saying "The method scales up to 4 cores." is not a reason! :)
- Make a plan of the things you want to investigate, it's easy to throw a lot of jobs at the cluster. Instead, estimate how long time all the jobs will take. Say, if you want to test for N core configurations, P parameters and each run takes 4 min on average, that amounts to $4NP$ minutes. You only have a limited amount of time to conduct experiments, AND you also have to write the report. Planning is key here!
- Labels on figures, and explanatory figure captions! (purposefully duplicated!)

Upload

- your report in PDF format
- a ZIP files with the Python sources, that are needed to run your code