MPI: Practical session 5

1 Mandelbrot set

Consider the recurrence relation

$$z_{n+1} = z_n^2 + z_0$$

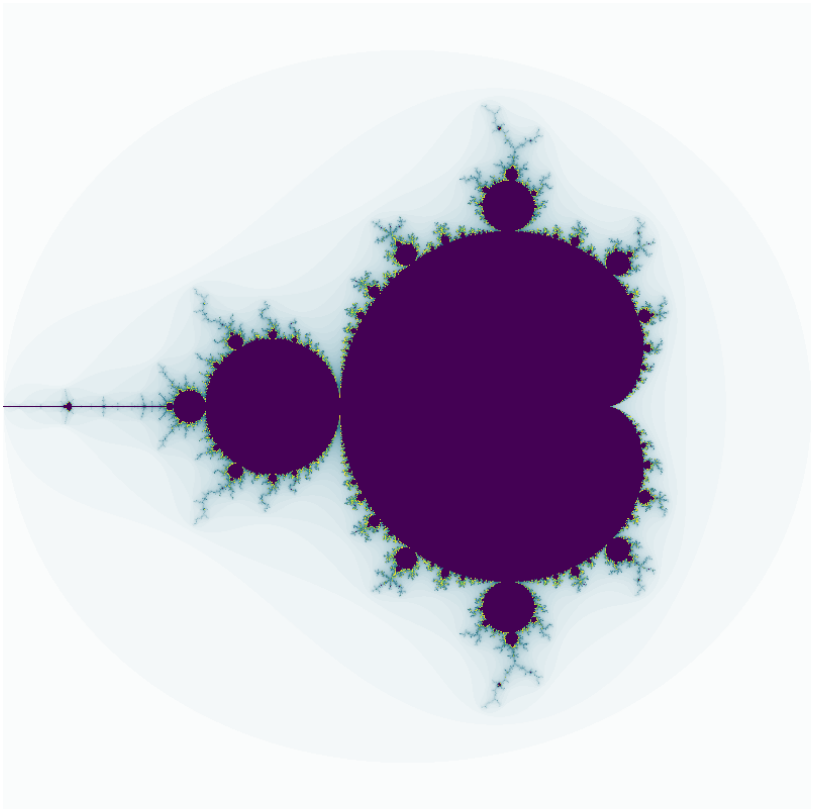
The Mandelbrot set is defined as the values of z_0 in the complex plane for which z_∞ is finite.

It can be shown that if for some n, $|z_n| > 2$, then z_∞ is finite. So we can iterate the above formula a few hundred times to see if $|z_n| > 2$.

The code in `mandelbrot.cpp` does exactly this. For each cell in a 2-dimensional domain: it iterates this formula `iters` times and stores the first n such that $|z_n| > 2$ (or `iters`, if no such n is found). These indices are then converted into RGB colour values according to a linear interpolation colour-map, and the results are output in a pam file (row by row). The pam file can be opened in Linux using:

```
$ open <file>
```

For example, if we run the program with a resolution of 800, domain $[-2, 1] \times [-1.5, 1.5]$ and 320 iterations, we get the following picture:



1.1 Mandelbrot in parallel

There are many ways to parallelise this code using MPI. As a first attempt, divide the domain horizontally in `nproc` blocks, each to be calculated by one process. At the end of the calculation, make each process send their results to rank 0 so that it can write the output file (same applies for the user-defined input).

1.2 Load balancing

This parallelisation is not yet optimal - work is not divided equally amongst the parallel tasks. This is because, for points inside the Mandelbrot set, the loop needs to execute `iters` times, whereas in the exterior, it might exit after a couple of iterations.

Modify your program to improve load balancing, while still dividing the domain into (approximately) equal number of rows for each process.

1.3 Master-slave parallelisation

To get even better load balancing, implement a master-slave approach, where rank 0 sends work to the other processes dynamically.

Let each slave keep count of the number of rows it has calculated, and print this, together with its rank, immediately before exiting. Are they all the same? Are they all the same if the resolution is not divisible by the number of slaves? Does the algorithm adapt well when the resolution is not divisible by the number of slaves?

Compare the performance of the three parallel programs you wrote by running them on 1, 2, 4, 8, 16 processors and plotting a graph of

$$\frac{\text{Run-time on 1 processor}}{\text{Run-time on } P \text{ processors}}$$

against P. In order to provide better performance, consider removing the code that is used to output the data, so that (virtually) all of the run-time is spent doing useful computations.

1.4 Area calculation

The area of the Mandelbrot set is unknown there are merely numerical approximations to it, and some analytic upper and lower bounds. Modify the code so that each rank calculates the area of Mandelbrot set that it has been asked to calculate (and not the set itself), and then sum these with an `MPI_Reduce`.

(The accepted answer is in the region of 1.506592. We would expect a slight bias to overestimating, for the mathematical definition of z_∞ being finite is stricter than our definition of $|z|$ not exceeding two after `iters` iterations.)

1.5 Fun with the Mandelbrot

To create even more impressive images of the Mandelbrot, you can use this this Mandelbrot gallery, which provides both the coordinate data for the domain and the maximum number of iterations used to create the images. See replicated examples in the next page

Feel free to explore the Mandelbrot set on your own. You can change the colour scheme by changing the RGB values of `colors` in the code. As you zoom in further, it can be useful to increase the maximum number of iterations. However, eventually the precision of double precision arithmetic becomes an issue, and the set stops appearing fractal, but becomes smoothed out.

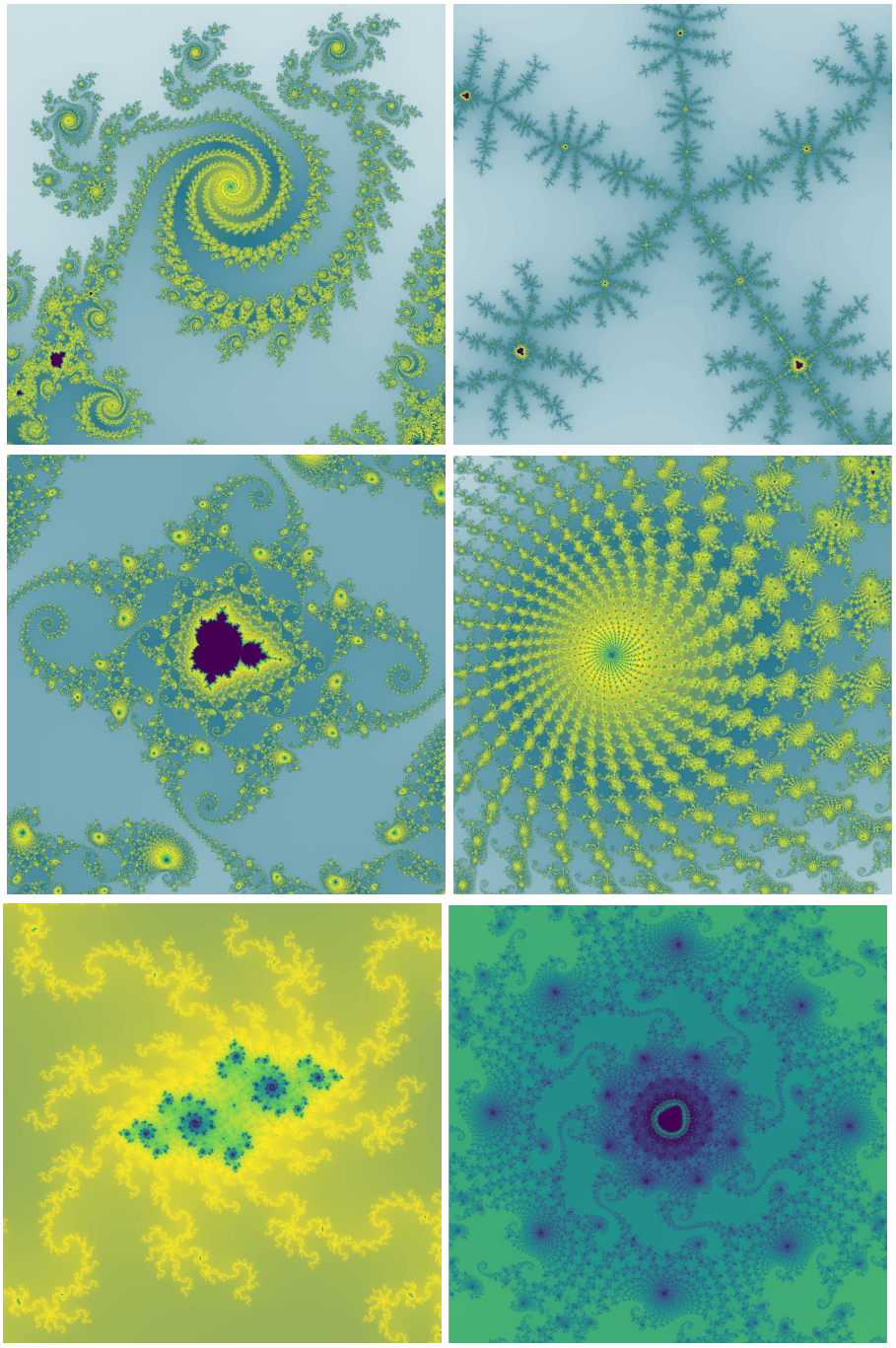


Figure 1: Images of the Mandelbrot set (parameters from https://www.sci-pi.org.uk/mandel/mandel_gallery/)