Department of Electronic Systems

Numerical Scientific Computing (NSC)

MINIPROJECT

Computation of the Mandelbrot set using different approaches in Python

---

Author:
Tor Kaufmann Gjerde

Work performed during the NSC course in the period March - May 2021

1 Software

A set of different software implementations are developed in order to get experience with different methods for computation and optimizing for scientific computation. Python is used as the main language for developing the implementations. The Mandelbrot set is computed with the following approaches.

- A naive version only implementing the standard python language. In the project file code folder this implementation is found as the file:
  naive_mandelbrot.py
- A version with Python Numpy functionality. This is implemented in the following file:
  numpy_mandelbrot.py
- A multicore parallel version using the Python multiprocessing package. Implemented in the file:
  multicore_mandelbrot.py
- A multicore parallel version using the dask distributed package for Python. Implemented in the file:
  dask_parallel_mandelbrot.py
- A Cython version where Python is compiled into C code. This implementation is found in the folder **cython_mandelbrot** under the file: **cython_naive_mandelbrot.py**.
- A Graphic Processing Unit (GPU) - accelerated version using the PyOpenCL package. In the project code folder this version is implemented in the files: **GPU_mandelbrot_kernel.cl** and **GPU_mandelbrot.py**

All files with the **.py** extension in the above bullet points are built to fully run the Mandelbrot computation separately. They all contain a function plotting functionality as well as a crude user input option from terminal in order to choose plotting and save output data. All implementations are tested and produce a correct plot of the Mandelbrot set. The GPU version with PyOpenCl would probably need to be rebuilt for running on specific hardware other than that tested in this mini-project.

Function input, output and overall functionality is described and commented in code and should be fairly easy to read and understand.

1.1 Mandelbrot Algorithm

The basic Mandelbrot algorithm used for all implementations in this mini-project is presented in the following.

Algorithm 1: Mandelbrot map algorithm

Result: Mapped array
for *n in Size* **do**
    $c = complex(Re, Im)$;
    $z = complex(0, 0)$;
    for *i in Iterations* **do**
        $z = z^2 + c$;
        if $abs(z) > Threshold$ **then**
            $mapped = abs(z)/Iterations$;
        end
        if $i == Iterations$ **then**
            $mapped = 1$;
        end
    end
end

Algorithm 1 Shows how for two input arrays, one containing the real and the another containing the imaginary component of a complex number - the resulting complex number is quadratically mapped to either 1 or lower.

Some slight variations for the input type of the algorithm is observed throughout the different Python scripts supplied in this mini-project. For the multiprocessing versions the complex number input is supplied as a single variable and not two arrays or variables each containing real and imaginary components.

1.2 Testing and Designing

During the design and building of this software, plotting of the generated Mandelbrot sets using the python package **matplotlib** has been used for validating the correctness of implementation.

The name "map" is used as an extension to different variables and function names throughout the code and hints to the process of mapping the Mandelbrot set to a set of complex numbers in a coordinate system. Real component are interpreted along the first axis (x) and Imaginary components along the second axis (y).

During testing and design of the software a lot of observations specifically related to execution time was done. For the Dask implementation the downloading and uploading of the complex matrix ie. input data to the client seemed to be the biggest bottleneck - as this clearly took more time than the algorithm computation itself when the amount of workers increased to above 2. More time could be used on exploring the possibilities of doing more operations on the cluster as well as fully implementing dask arrays. However this does not get away with the problem of gathering the complete mapped matrix down to the local process for data plotting.

Other observations included the size of different data formats. The Numpy implementation was praised for being relative fast for saving output data to h5py files. Large Python lists with output data was deemed to big to save in a reasonable amount of time.

1.3 Profiling and bench-marking

As a way of testing and comparing the different implementations a test scheme was developed. Common parameters like output matrix size, and Mandelbrot algorithm parameters were held constant across all implementations and the Python *time.time*() function was used to measure the algorithm computation time specifically for each implementation.

The testing was done with the following parameters held constant:

- Output square matrix dimension = 5000 = $25 * 10^6$ computational elements.
- Mandelbrot Threshold = 2
- Mandelbrot Iterations = 200

The tests were performed on a workstation platform with the following specifications:

- Processor: AMD Ryzen 5 2600 six-core processor x 12
- Graphics: AMD Radeon (tm) rx 480 graphics
- OS: Ubuntu 20.04.2 LTS
- OS Type: 64-bit
- Memory: 15.6 GiB
- Disc capacity: 360 GB

The accelerated GPU version was run on an Apple Macbook pro due to GPU comparability issues: Intel(R) Iris(TM) Graphics 6100' on 'Apple' Macbook Pro Retina A2015

The following table present the timing obtained for the different implementations.

	Naive	**Numpy**	**Multicore (2 W)**	**Multicore (12 W)**	**Dask**	**Cython**	**GPU**
Time(minutes, seconds)	>30 min	>25 min	8 min 28 sec	2 min 27 sec	n/a	2 min 9 sec	2.5 sec

As we can see the GPU accelerated version with PyOpenCl clearly comes out as the winner in terms of raw computational speed. Another observation is that Cython comes out at a faster speed than multiprocessing even at 12 cores for this specific implementation.

The following figure gives an overview of the speed-up observed for an increase in the amount of workers for the Python multiprocessing implementation. The amount of workers was increased from 1 up to the available 12 in an increment of 2.

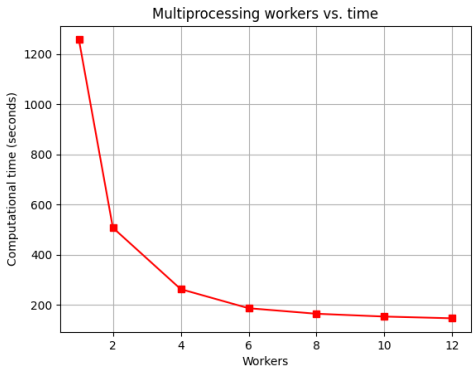

Figure 1

1.4 Readme file

This document can be interpreted as an extended readme file. The document and project folder containin all code is also available in the online GitHub repository at:
https://github.com/TORKGB/Mandelbrot_NSC_miniproject

2 Output Mandelbrot set Plot

Mandelbrot set plots generated from the GPU accelerated implementation.

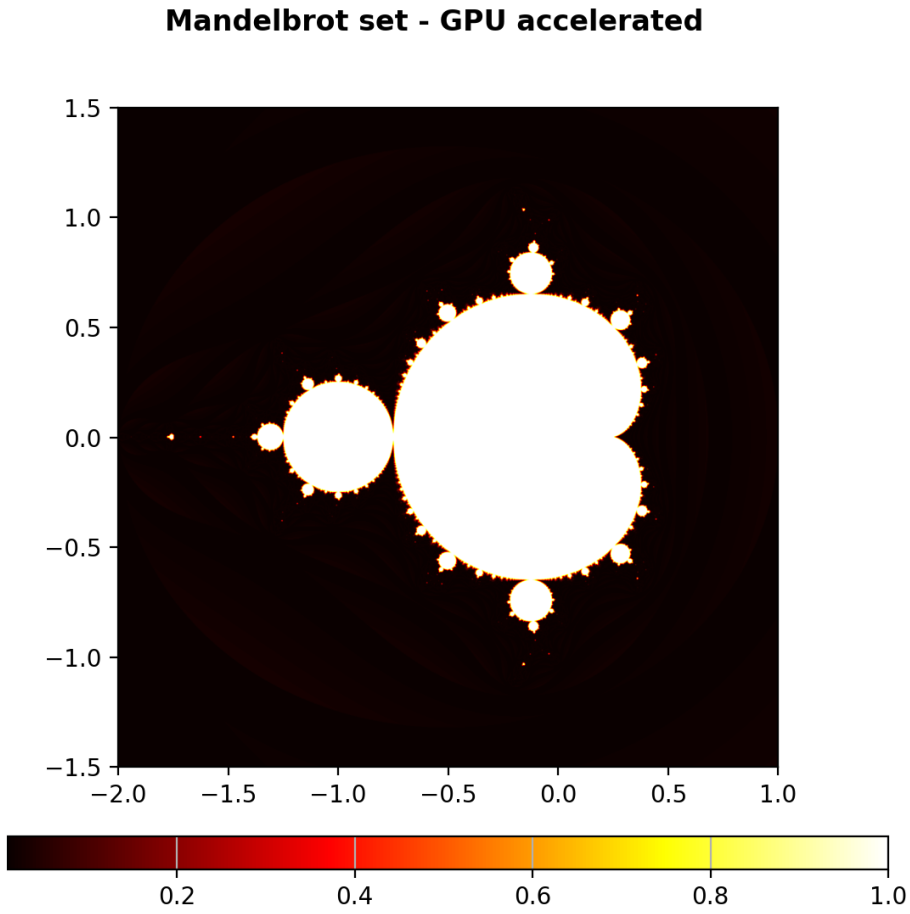

Figure 2: Mapped output matrix of size 5000 by 5000

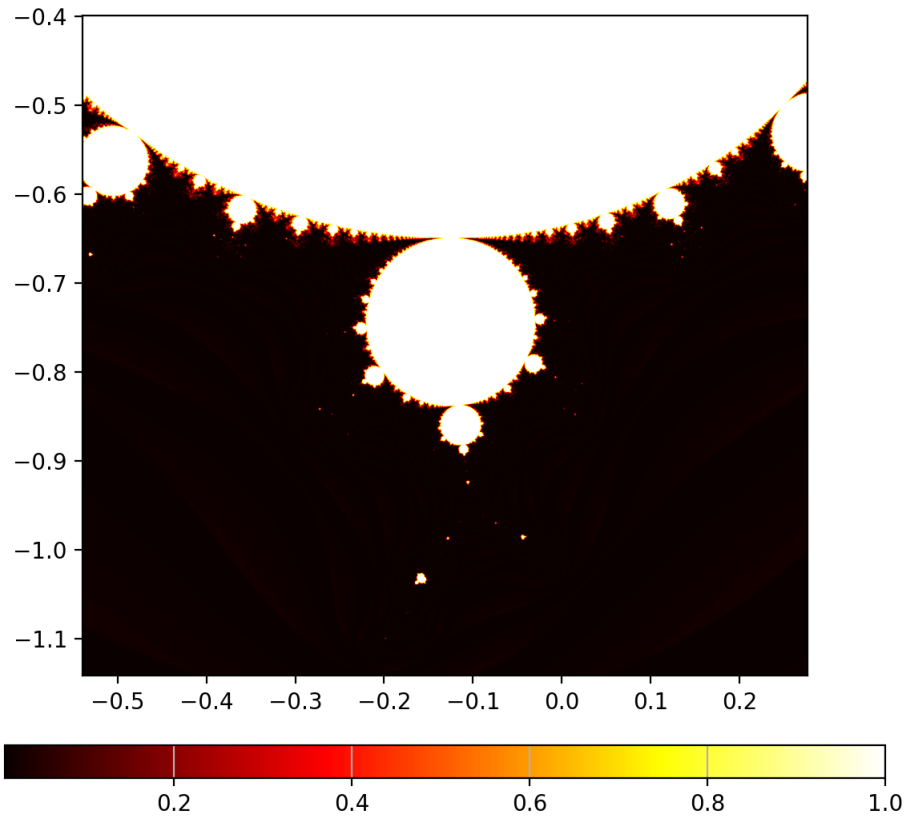

Figure 3: Mapped output matrix of size 10000 by 10000