Mandelbrot Set – Part 3

Numerical Scientific Computing Mini-Project

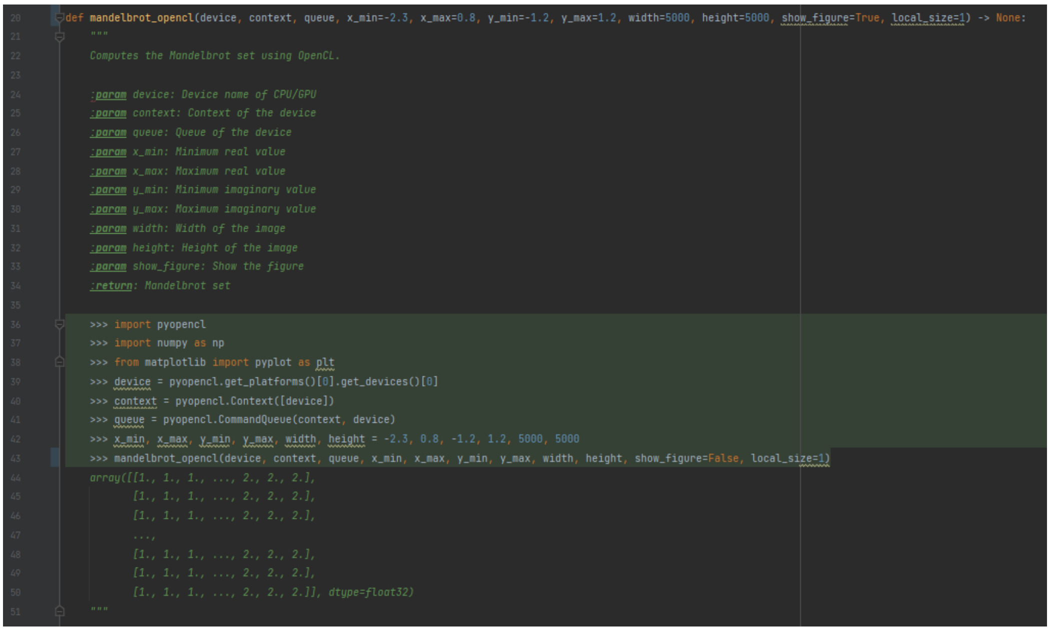
Name: Lukas Bisgaard Kristensen

Date: 26/04/2023

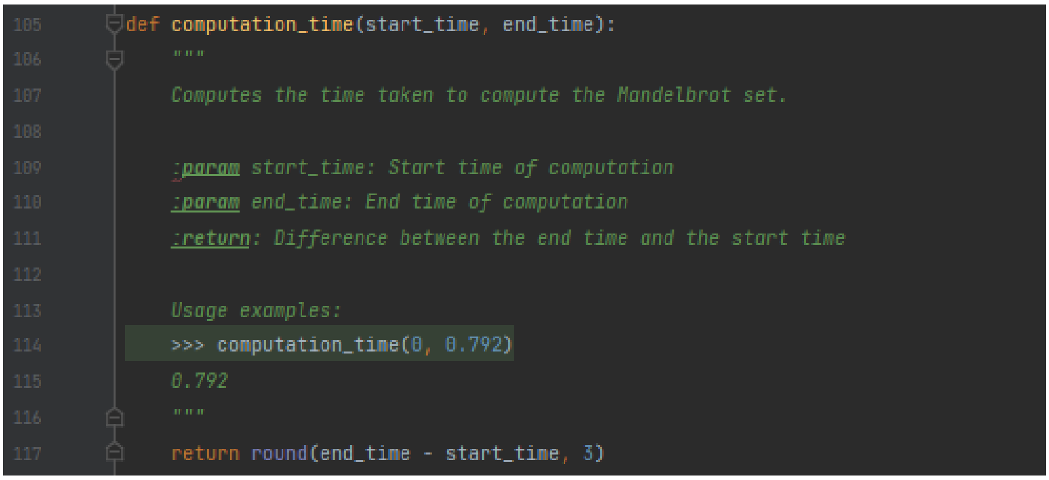
Program: Computer Engineering (AVS), 8th semester, Aalborg University

Course: Numerical Scientific Computing

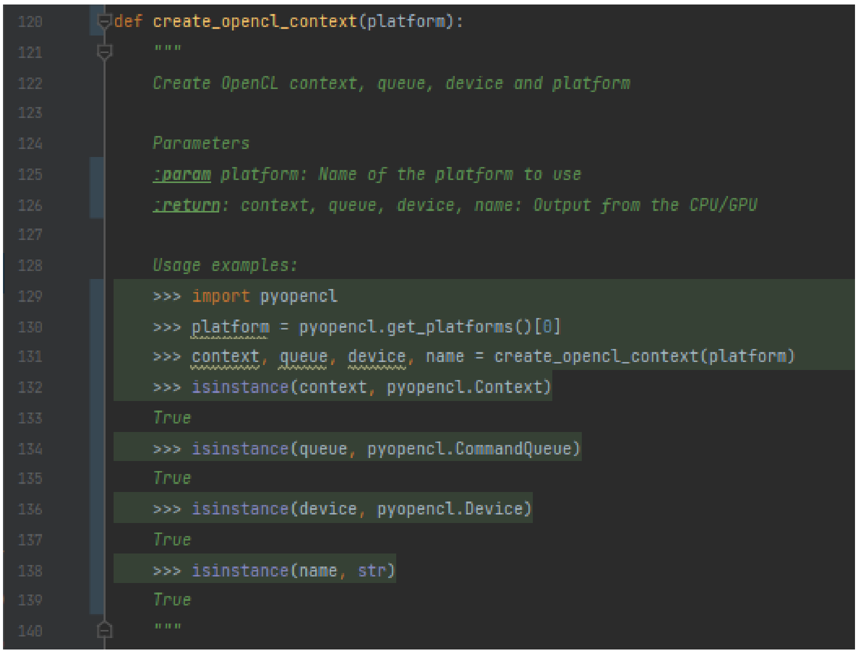
Docstrings and Doctests (3 cases), see mandelbrot_opencl.py



```python
def mandelbrot_opencl(device, context, queue, x_min=-2.3, x_max=0.8, y_min=-1.2, y_max=1.2, width=5000, height=5000, show_figure=True, local_size=1) -> None:
    """
    Computes the Mandelbrot set using OpenCL.

    :param device: Device name of CPU/GPU
    :param context: Context of the device
    :param queue: Queue of the device
    :param x_min: Minimum real value
    :param x_max: Maximum real value
    :param y_min: Minimum imaginary value
    :param y_max: Maximum imaginary value
    :param width: Width of the image
    :param height: Height of the image
    :param show_figure: Show the figure
    :return: Mandelbrot set

    >>> import pyopencl
    >>> import numpy as np
    >>> from matplotlib import pyplot as plt
    >>> device = pyopencl.get_platforms()[0].get_devices()[0]
    >>> context = pyopencl.Context([device])
    >>> queue = pyopencl.CommandQueue(context, device)
    >>> x_min, x_max, y_min, y_max, width, height = -2.3, 0.8, -1.2, 1.2, 5000, 5000
    >>> mandelbrot_opencl(device, context, queue, x_min, x_max, y_min, y_max, width, height, show_figure=False, local_size=1)
    array([[1., 1., 1., ..., 2., 2., 2.],
           [1., 1., 1., ..., 2., 2., 2.],
           [1., 1., 1., ..., 2., 2., 2.],
           ...,
           [1., 1., 1., ..., 2., 2., 2.],
           [1., 1., 1., ..., 2., 2., 2.],
           [1., 1., 1., ..., 2., 2., 2.]], dtype=float32)
    """
```



```python
def computation_time(start_time, end_time):
    """
    Computes the time taken to compute the Mandelbrot set.

    :param start_time: Start time of computation
    :param end_time: End time of computation
    :return: Difference between the end time and the start time

    Usage examples:
    >>> computation_time(0, 0.792)
    0.792
    """
    return round(end_time - start_time, 3)
```

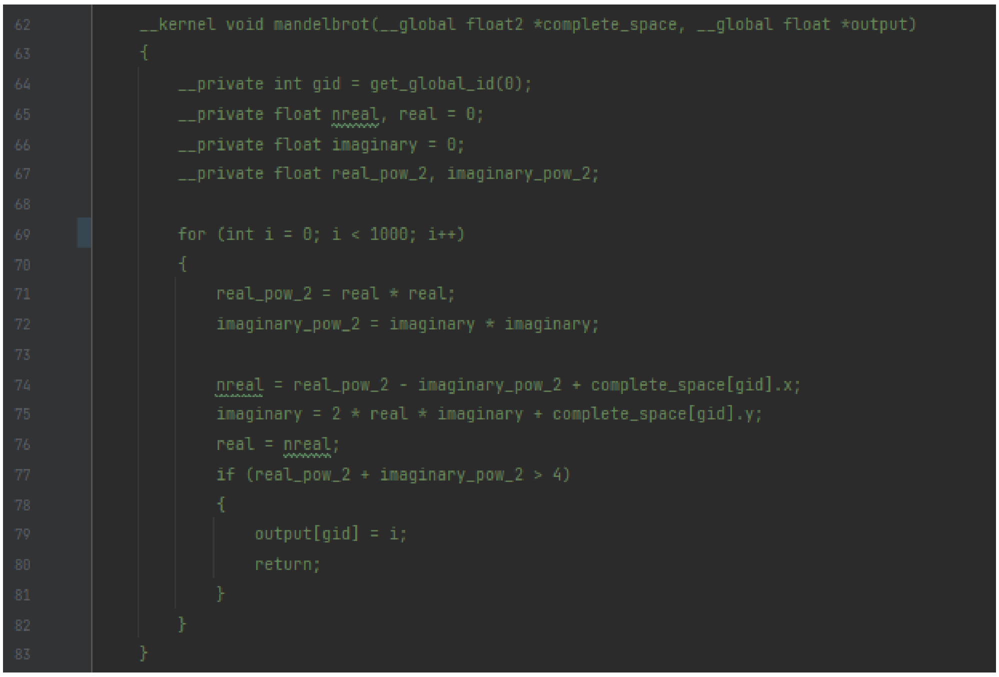


```
def create_opencl_context(platform):
    """
    Create OpenCL context, queue, device and platform

    Parameters
    :param platform: Name of the platform to use
    :return: context, queue, device, name: Output from the CPU/GPU

    Usage examples:
    >>> import pyopencl
    >>> platform = pyopencl.get_platforms()[0]
    >>> context, queue, device, name = create_opencl_context(platform)
    >>> isinstance(context, pyopencl.Context)
    True
    >>> isinstance(queue, pyopencl.CommandQueue)
    True
    >>> isinstance(device, pyopencl.Device)
    True
    >>> isinstance(name, str)
    True
    """
```

OpenCL with defined memory types for all variables

__global memory data type for the input and output data.

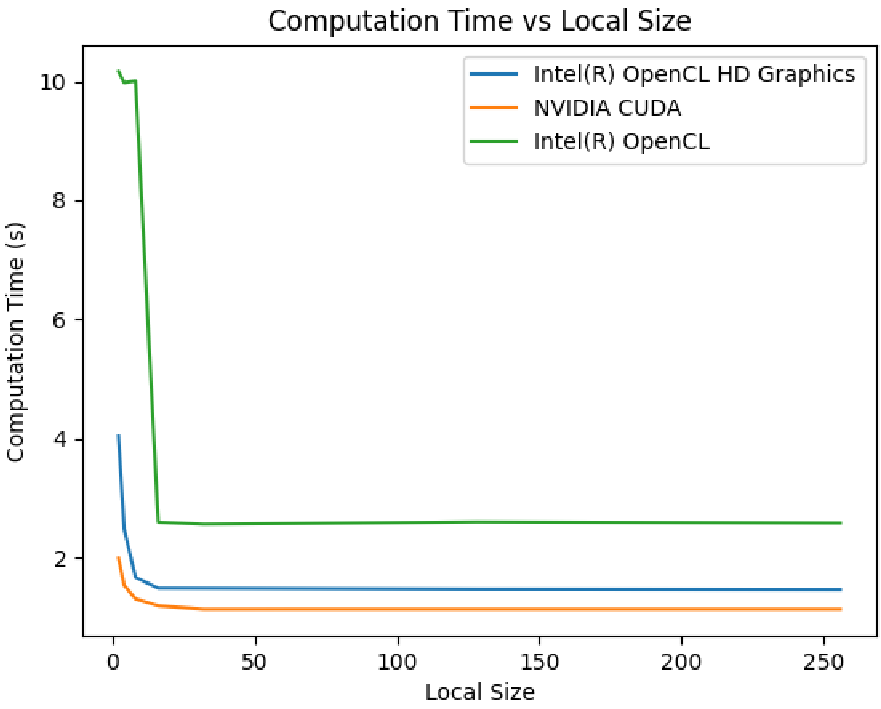
__private memory data type for data that is only relevant for workers within the function.



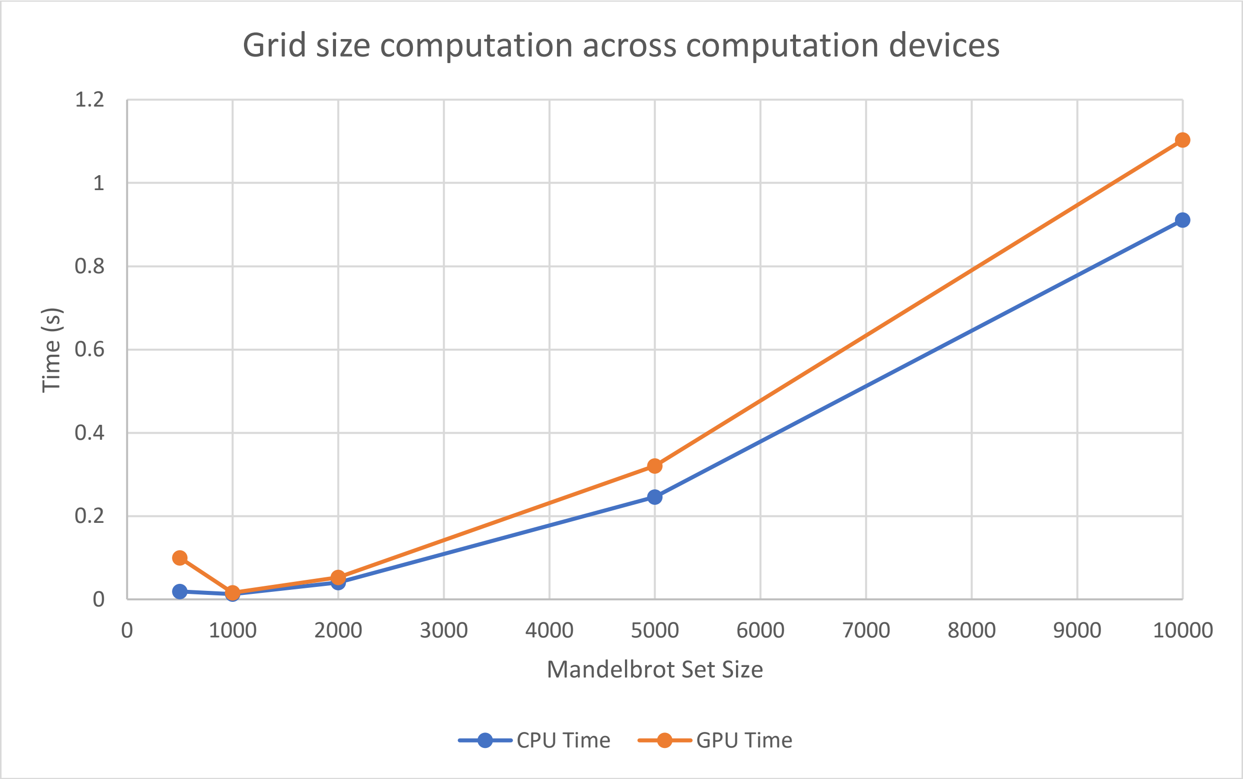
```
__kernel void mandelbrot(__global float2 *complete_space, __global float *output)
{
    __private int gid = get_global_id(0);
    __private float nreal, real = 0;
    __private float imaginary = 0;
    __private float real_pow_2, imaginary_pow_2;

    for (int i = 0; i < 1000; i++)
    {
        real_pow_2 = real * real;
        imaginary_pow_2 = imaginary * imaginary;

        nreal = real_pow_2 - imaginary_pow_2 + complete_space[gid].x;
        imaginary = 2 * real * imaginary + complete_space[gid].y;
        real = nreal;
        if (real_pow_2 + imaginary_pow_2 > 4)
        {
            output[gid] = i;
            return;
        }
    }
}
```

Local grid sizes (work group)

All CPU, GPU and integrated GPUs increase in performance as the local size increases.
The Intel® OpenCL HD Graphics is the integrated GPU and the Intel® OpenCL is the CPU. Here it can be seen that the integrated GPU significantly outperforms the CPU.



Global grid size (mandelbrot size)



Extra features

Zoom Animation (Generates a video output, see video)

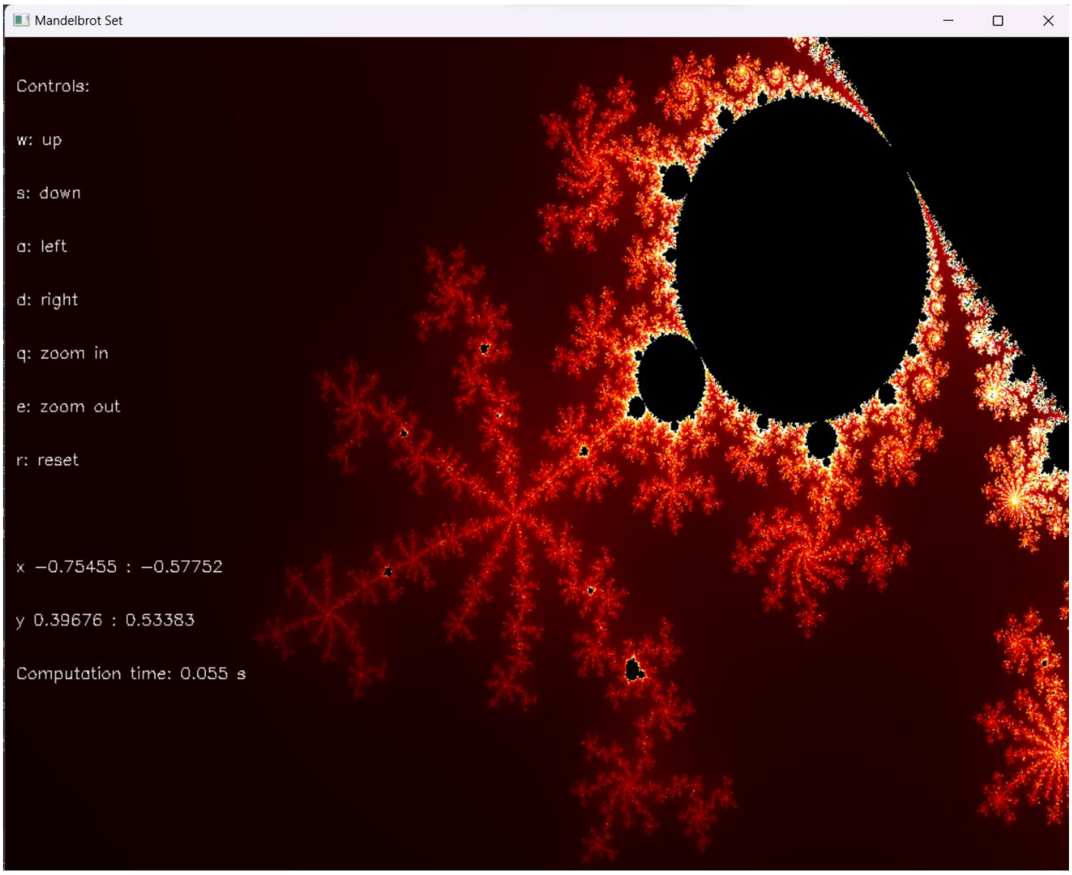
- Path to code: “Extra Features/mandelbrot_iteration_animation.py”
- YouTube Video of generated output: https://www.youtube.com/watch?v=L2zKIrriDfI

Iteration Animation (Generates a video output, see video)

- Path to code: “Extra Features/mandelbrot_animation.py”
- YouTube Video of generated output: https://www.youtube.com/watch?v=8Bjqgaluses

Mandelbrot Navigator (Interactive keyboard navigation)

- Path to code: “Extra Features/mandelbrot_navigator.py”



```
Controls:

w: up

s: down

a: left

d: right

q: zoom in

e: zoom out

r: reset


x -0.75455 : -0.57752

y 0.39676 : 0.53383

Computation time: 0.055 s
```