Docstring and Unit testing:

For each file, naive_approach, numpy_approach, numba_approach, multiprocessing_approach docstring parts have shown in the python files.

1. For naive_approach -> doctest implemented
2. For numpy_approach -> unittesting implemented
3. For numba_approach -> pytest implemented
4. For multiprocessing_approach -> unittesting implemented

– All the test cases worked perfectly

Mandelbrot Opencl execution:

Script: mandelbrot_opencl.py, mandelbrot_opencl_cpu.py
Kernel File: mandelbrot_opencl.cl

Results Summary

1. Performance on Intel i5-8250U CPU using Portable Computing Language:
    - Platform: Portable Computing Language
    - Device: pthread-Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz
    - Execution Times:
        - Width 500: 0.0955 seconds
        - Width 1000: 0.1236 seconds
        - Width 1500: 0.1275 seconds

2. Performance on NVIDIA GeForce RTX 3070 Laptop GPU:
    - Platform: NVIDIA CUDA
    - Device: NVIDIA GeForce RTX 3070 Laptop GPU
    - Execution Times:
        - Width 500: 0.0104 seconds
        - Width 1000: 0.0100 seconds
        - Width 1500: 0.0 seconds (indicating a very rapid calculation)

3. Performance on AMD GPU (gfx1035):
    - Platform: AMD Accelerated Parallel Processing
    - Device: gfx1035
    - Execution Times:
        - Width 500: 0.0 seconds (indicating a very rapid calculation)
        - Width 1000: 0.0027 seconds
        - Width 1500: 0.0 seconds (indicating a very rapid calculation)

Analysis

1. GPU vs. CPU Performance: When it comes to processing the Mandelbrot set, both the NVIDIA and AMD graphics cards were way faster than the Intel CPU. This is because the graphics cards have thousands of small processing units that can work together to handle many calculations at once. The CPU, on the other hand, has fewer processing cores which means it can’t handle as many tasks at the same time.

2. Comparing the Two GPUs: Between the two graphics cards, the NVIDIA RTX 3070 was consistently a bit faster than the AMD card. This could be because the NVIDIA card is built on a newer design and has better processing capabilities, which seems to give it an edge in quickly crunching numbers for our fractal patterns.

3. How the CPU Held Up:Even though the CPU was slower, it still did a pretty good job. It shows that the tool I used, PyOpenCL, is really good at making the most out of whatever processor it has to work with. It can use every part of the CPU effectively to still get the job done.

I also learned that the way a processor handles memory is super important. For instance, graphics cards can use something called local memory to speed things up. This is a faster type of memory that's shared among processing units working on the same problem. If they can share data quickly, they don’t waste time waiting around for it, which speeds up the whole process.