Numerical Scientific Computing

Jakob Lund Jensen
jjens19@student.aau.dk

November 29, 2024

Mini Project Description

1 Title

Mini-project 3

2 Moodle description

For this hand-in, you are expected to deliver your new work on the Mandelbrot mini-project, including the following:

Code as runnable Python files:

1. one or more updated versions of your previous code (naive, numpy or multiprocessing) where you add the following elements:
   (a) Docstrings for two or more functions, explaining purpose of function, and input/output variables.
   (b) Unit testing using one of the presented frameworks. You need to include at least 3 test cases.
2. OpenCL implementation of the Mandelbrot algorithm, including benchmarking results for different grid sizes, comparing multiple available compute devices, e.g., CPU and GPU. Also, proper use of different memory types (global, constant, local, private) should be considered.

Worksheet (either as supplementary PDF or as executable Jupyter Notebook) that includes a comparison of performance (in terms of execution time) for each of the algorithms as well as considerations on selection of memory type for different variables.

3 Docstrings

Two docstrings have been written and can be found inside of the *MiniProject3Numpy.py* file. These docstrings describe the function *Mandelbrot()* and *complex_matrix()*. The docstrings were written using the structure found in the *example_docstring.py* example from lecture 8. This structure consists of giving a brief explanation of the function, followed by an overview of the input and output.

4 Unit testing

Three unique unit tests were constructed using the unittest framework and can be found in *MiniProject3Unittest.py*. The purpose of these tests is to test whether the functions *Mandelbrot()* and *complex_matrix()* work as intended. The three tests are:

1. test_Mandelbrot_set
   Purpose: Test if the *Mandelbrot()* function gives the correct output.
   Execution: The function was executed using a complex matrix with a pixel density of 5 (a 5x5 matrix). Then the output was compared to an output matrix that had been calculated manually.
   Result: Passed

2. test_complex_matrix
   Purpose: Test if the *complex_matrix()* function gives the correct output.
   Execution: The function was executed using a pixel density of 5. Then the output was compared to a matrix that had been calculated manually.
   Result: Passed

3. test_size
   Purpose: Test whether the output size of the *Mandelbrot()* function is the same size as the input.
   Execution: The size of the output matrix of the *Mandelbrot()* function was compared to the output matrix of the *complex_matrix()* function, which is used as the input for the *Mandelbrot()* function.
   Result: Passed

5 OpenCL implementation

The OpenCL implementation can be found in the *MiniProject3OpenCL.py* file.

Usage of memory types:

- Global: No global variables were used, as they were not necessary. Using global variables improperly would only slow down the program.

- Constant: The complex matrix used for computing the Mandelbrot set was called as a constant variable within the kernel_code. This was done as the variable is a read-only variable. With proper implementation, this memory type can be twice as quick as a normal local type.
- Local: All other variables are local variables. While some variables are being used as read-only, these are only called a couple of times, leading to the extra execution time being insignificant. Leaving them as local variables also increases the flexibility while further working or debugging the code, as they maybe be used.
- Private: No private variables were used.

The following benchmark tests were executed on my PC, which has an NVIDIA Quadro P520 GPU and an INTEL UHD Graphics 620.
Benchmark tests were performed for different pixel densities, grid sizes/sizes of work groups for two different GPU devices. I got a little lazy with generating the graphs, but here is some explanation.

- 6 graphs were generated based on 6 different image sizes.
- The Mandelbrot set was generated using a threshold of 100 for all 6 graphs.
- Graphs were made for 1000x1000, 2000x2000, 6000x6000, 8000x8000, 10000x10000, and 12000x12000 pixels.
- Worker index is for the size of the work group. The different sizes used were (1,1), (4,4), (8,8), (10,10), (20,20), and (25,25).
- The time is in seconds.
- The blue line is my NVIDIA GPU, while the orange line is my INTEL GPU.
- My NVIDIA GPU ran out of memory after 8000x8000.
- My INTEL GPU would crash when using (20,20) or (25,25) work groups at less than 10000x10000. The crash may be a result of a bug in the code, however, I was not able to locate anything suspicious.

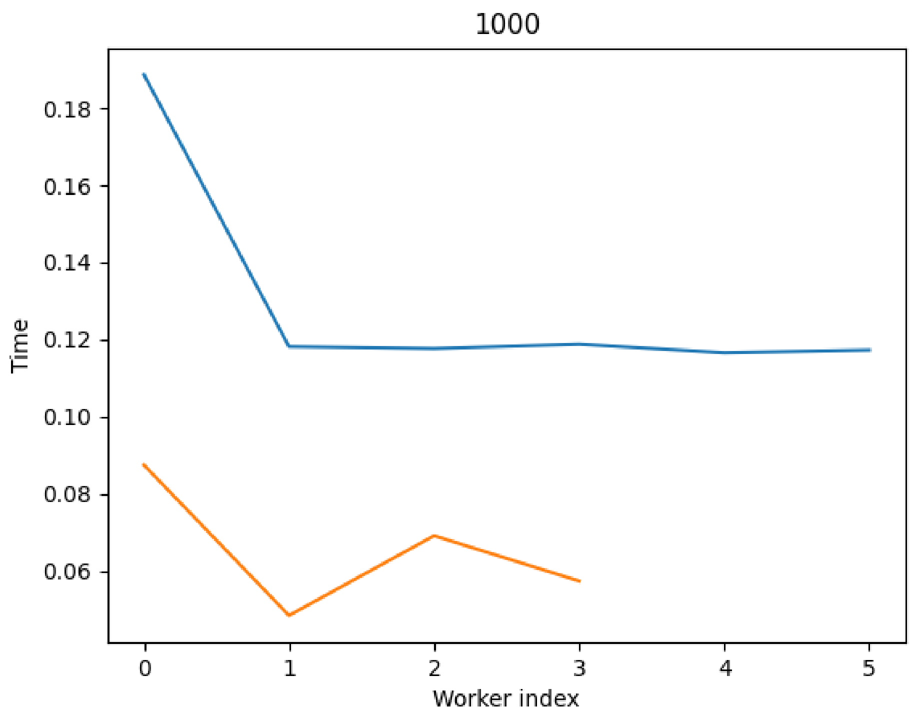

Figure 1: Benchmark test for 1000x1000

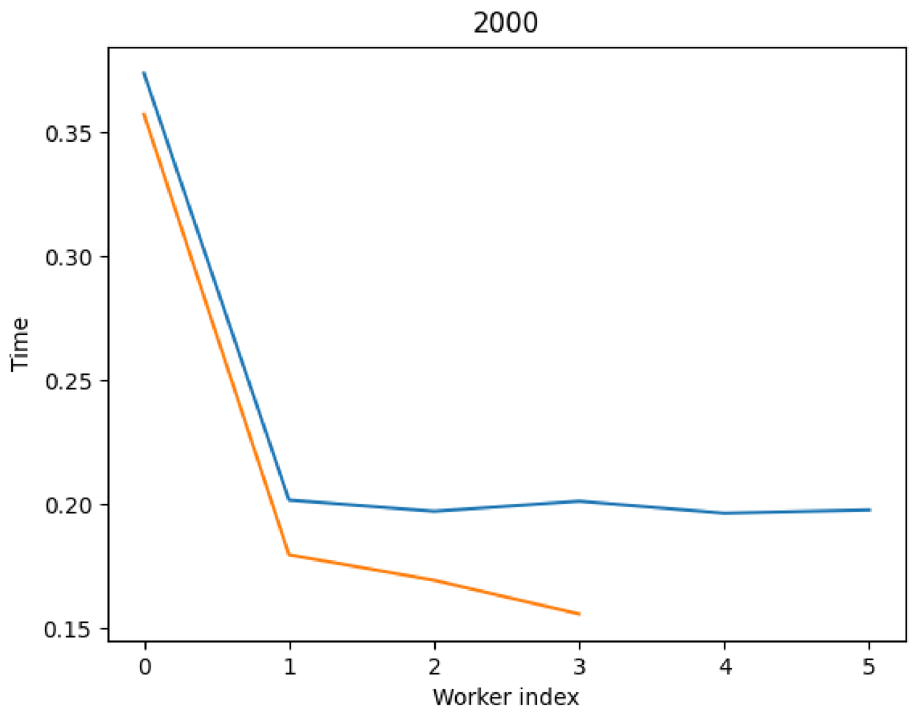

Figure 2: Benchmark test for 2000x2000

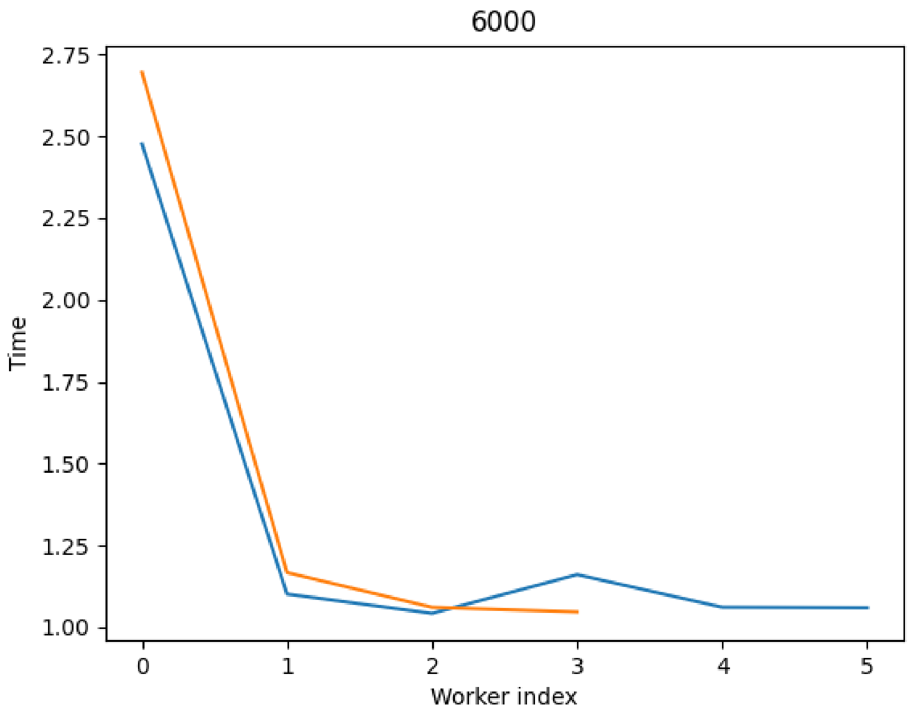

Figure 3: Benchmark test for 6000x6000

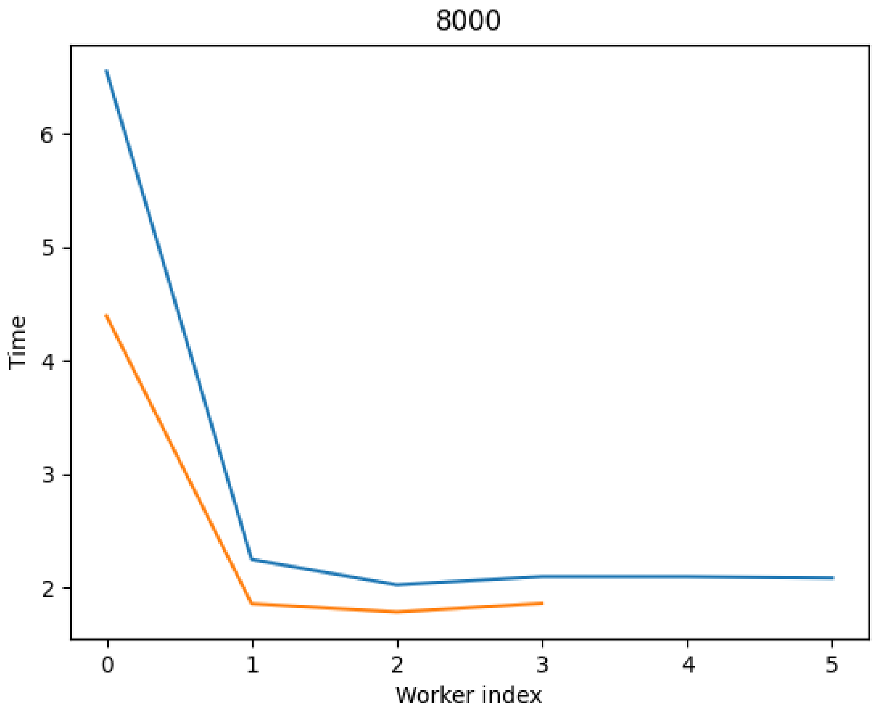

Figure 4: Benchmark test for 8000x8000

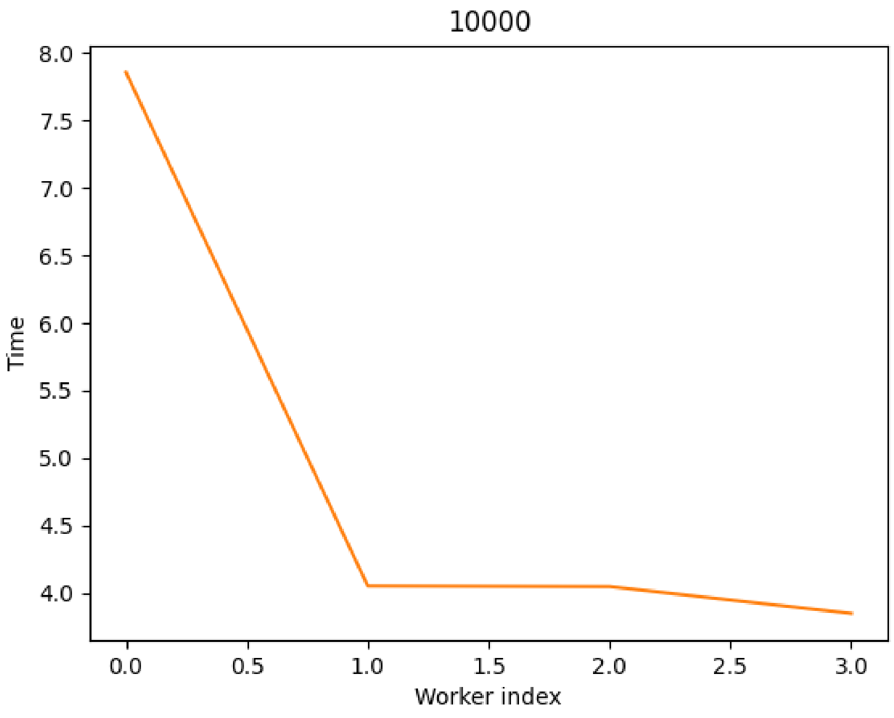

Figure 5: Benchmark test for 10000x10000

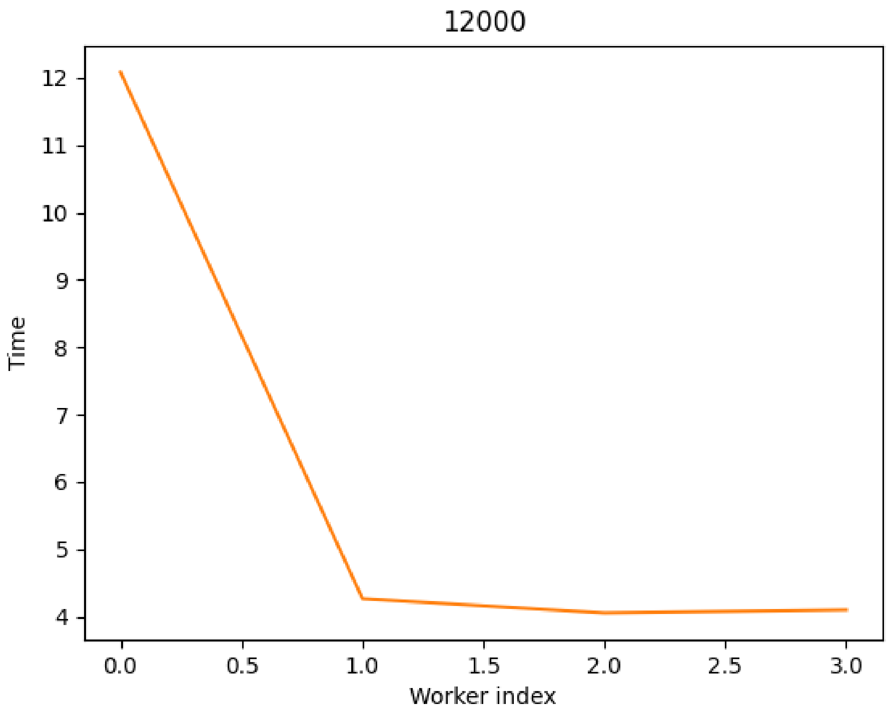

Figure 6: Benchmark test for 12000x12000