# Report – Exercise two

Since both of us haven't had lots of math during our bachelor's degree, we had a lot of trouble with the exercise, since the understanding wasn't there regarding the complex plane. Therefore, we searched a lot about this topic and tried to come up with a working solution. So firstly, we created a C++ project. To use OpenMP, we hadn't to import anything else. Since our knowledge is very, very basic we used the following guides:

- MPI vs OpenMP: A case study on parallel generation of Mandelbrot set[1]
- Let's draw the Mandelbrot set![2]

For the calculation of the performance we used chrono::steady_clock::now().

Our approach was to build two different solutions so we can derive the parallel solution from the serial solution, so that we can measure the performance of the solutions easier. Then we read in the image size through the command line (w,h). To compute every pixel, we created two for loops. One for the row, one for the column. Then we calculated the transformed real and imaginary part and evaluate the points for the set.

Since we never worked in detail with C++, we searched for a way to produce an image and come across EasyBMP[3]. With this class and the documentation, it was very easy to build an image.

For the sequential part we tried the following inputs:

| Width | Height | Time |
|---|---|---|
| 2000 | 2000 | 2444.73 ms |
| 4000 | 4000 | 9830.59 ms |
| 1000 | 1000 | 617.852 ms |

For the parallel part, we thought about what we can do in parallel. For our understanding at the moment, the perfect part to parallel would be the following code snippet.

```
while ((x*x+y*y) < 4 && currentIteration < MAX_ITERATIONS) {
    double x_new = x*x-y*y+c_re;

    y = 2*x*y+c_im;
    x = x_new;

    currentIteration++;
}
```

This is because the calculation of the points can be done on its own, the image just needs to know, how to colorize each pixel.

With re-writing the code, we realized, that we had to change the CMakeLists.txt file to enable OpenMP, which is funny, because CLion and/or the project seems to know the Pragma definition.

[1] See https://revistas.ulasalle.edu.pe/innosoft/article/view/29
[2] See https://jonisalonen.com/2013/lets-draw-the-mandelbrot-set/
[3] See http://izanbf.es/project/easy-bmp/

With adding the line

```
#pragma omp parallel default(none) private(c_re, c_im, x, y)
shared(currentIteration)
```

for the block above, we receive the following output (with a false result):

| Width | Height | Time |
|---|---|---|
| 1000 | 1000 | 33699.1 ms |

So, we definitely have an error in our code, therefore we don't add the other examples. With debugging the solution, we realized, that currentIteration is always 1000. It equals the MAX_ITERATIONS variable, which seems rather confusing. With a bit more refactoring and debugging we realized that the variables for the complex number is always 0. Therefore, the while loop is more or less a while(true) at the moment. With sharing the variables, it worked out somehow. What's not really working is the speed up. The parallel solution slowed the whole solution down.

| Width | Height | Number of Threads | Time |
|---|---|---|---|
| 1000 | 1000 | 6 | 32473.6 ms |
| 1000 | 1000 | 3 | 21815 ms |
| 1000 | 1000 | 2 | 3831.22 ms |
| 1000 | 1000 | 1 | 3814.94 ms |

So basically, we have now a working solution, but it is much, much slower than the serial solution. Another interesting fact is that when we increase the number of threads, the performance is also going down rapidly.

With running the for loops also in parallel we got a huge speed up:

| Width | Height | Number of Threads | Time |
|---|---|---|---|
| 1000 | 1000 | 6 | 1165.4 ms |
| 1000 | 1000 | 3 | 2174.7 ms |
| 1000 | 1000 | 2 | 3798.79 ms |
| 1000 | 1000 | 1 | 3810.96 ms |
| 1000 | 1000 | 11 | 890.111 ms |

With enough Threads we can get around 680 ms, which is also slower than the sequential solution.

With removing

```
#pragma omp parallel default(none) private(c_re, c_im, x, y)
shared(currentIteration)
```

from the code, we got finally a speed up:

| Width | Height | Number of Threads | Time |
|---|---|---|---|
| 1000 | 1000 | 6 | 400.276 ms |
| 1000 | 1000 | 3 | 452.527 ms |
| 1000 | 1000 | 2 | 613.733 ms |
| 1000 | 1000 | 1 | 610.612 ms |
| 1000 | 1000 | 11 | 268.793 ms |
| 1000 | 1000 | 26 | 137.105 ms |
| 1000 | 1000 | 50 | 96.746 ms |

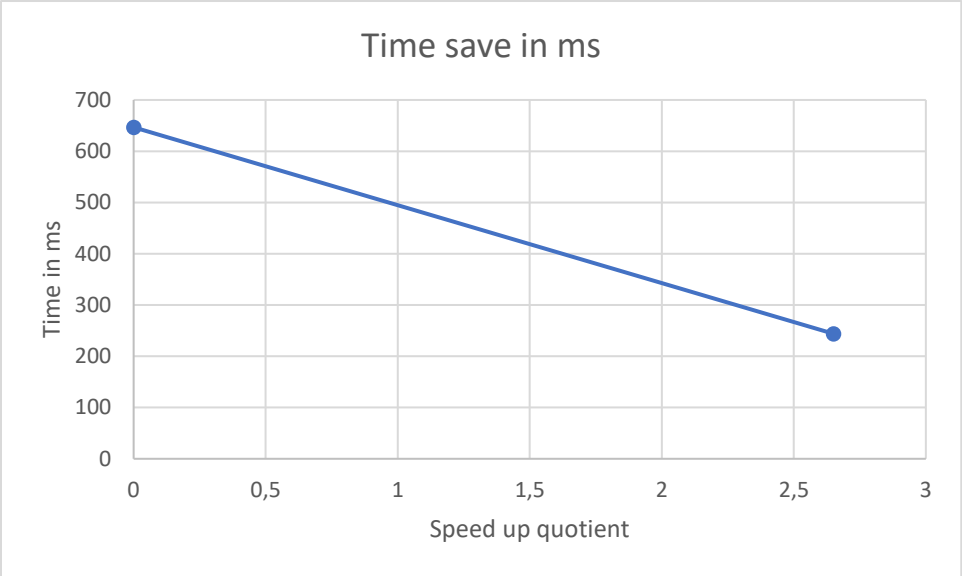
So, now that we got a first speed up, we would like to compare it to the same data as in our sequential solution. Since we got a really good speed up with 11 threads (resulted in the fact, that we didn't think about 0 as a thread), we will stick to 10 threads.

| Width | Height | Threads | Time |
|---|---|---|---|
| 2000 | 2000 | 10 | 1071.66 ms |
| 4000 | 4000 | 10 | 4299.88 ms |
| 1000 | 1000 | 10 | 272.642 ms |

Since it is already a speed up, we want finally to compare the question from the exercise sheet.

| Width | Height | Threads | Time |
|---|---|---|---|
| 1024 | 1024 | - | 646.65 ms |
| 1024 | 1024 | 12 | 243.967 ms |
| | | | |
| Speed up quotient | | | 2,65 |

Following the graph of the time save our my solution:




Final remarks:

We don’t know how it went for the others, but for us it was very difficult to really understand the assignment, because we have never dealt with this kind of mathematics before. Therefore, we lost a lot of time with doing research about the topic to get an idea of it. Nevertheless, we could finish the exercise and was able to get a decent speed up. We don’t think that this solution is optimized to the last bit, but we got an idea, how to deal with such an exercise. We wish, we had more time to deal with Open MP since we believe that we didn’t fully understand it. At least, we got a big warning block in CLion. When dealing with the warnings, we nearly crashed the whole application, so this would be a topic we would need to study more to use Open MP more efficiently. Since most of the iterations were between 1 – 10, we decided to map for the values 1-5 and everything else.