Lab 4 Report

Zhaoyi Wang 1689747

Part 1: FFI in Rust

Question 1

```
pub fn compute_euclidean_distance(p1: &Point, p2: &Point) -> f64 {
    // sqrt((x1-x2)^2 + (y1-y2)^2)
    return (((p1.x - p2.x) as f64).powi(2) + ((p1.y - p2.y) as
f64).powi(2)).sqrt();
}
```

Question 2

In `test.rs`:

```
#[test]
pub fn compute_eu_distance() {
    use super::*;

    let p1 = Point{
        x: 5,
        y: 6,
    };

    let p2 = Point {
        x: -7,
        y: 11,
```

```
    };

    let correct_result = 13.0;
    assert_eq!(compute_euclidean_distance(&p1, &p2), correct_result);
}
```

The output:

```
running 1 test
test test::compute_distance ... ok

test result: ok. 1 passed; 0 failed; 0 ignored; 0 measured; 0 filtered
out; finished in 0.00s
```

Question 3

```
use std::cmp::max;
pub fn compute_chebyshev_distance(p1: &Point, p2: &Point) -> i32 {
    let x_abs = (p1.x as i32 - p2.x as i32).abs();
    let y_abs = (p1.y as i32 - p2.y as i32).abs();
    max(x_abs,y_abs)
}
```

Question 4

In `main.rs` , we wil rewrite like this:

```
use std::cmp::max;
pub fn compute_chebyshev_distance_c(p1: &Point, p2: &Point) -> i32{
    unsafe {
        let x_abs = abs(p1.x as i32 - p2.x as i32);
        let y_abs = abs(p1.y as i32 - p2.y as i32);
        max(x_abs,y_abs)
    }
}
```

Question 5

`fn main()` as follows:

```
fn main() {
    println!("===== Distance Calculator =====");
    println!("Please input the coordinate for the 1st point.");
    println!("x1: ");
    let x1: i32 = handle_input();
    println!("y1: ");
    let y1: i32 = handle_input();
    let p1 = Point {
        x: x1 as i8,
        y: y1 as i8,
    };

    println!("Please input the coordinate for the 2nd point.");
    println!("x2: ");
    let x2: i32 = handle_input();
    println!("y2: ");
    let y2: i32 = handle_input();
    let p2 = Point {
        x: x2 as i8,
        y: y2 as i8,
    };

    println!("The points you entered are: ({},{}) and ({},{})",
             p1.x, p1.y, p2.x, p2.y
    );

    loop {
        println!("\nPlease choose what kind of distance you want to
get:");
        println!("1.Euclidean Distance \n2.Manhattan Distance
\n3.Chebyshev Distance \n4.Exit");
```

```
        let choice: i32 = handle_input();
        match choice {
            1 => {
                println!("Euclidean Distance is: {}",
compute_euclidean_distance(&p1, &p2));
            }
            2 => {
                println!("Manhattan Distance is: {}",
compute_manhattan_distance(&p1, &p2));
            }
            3 => {
                println!("Chebyshev Distance is: {}",
compute_chebyshev_distance(&p1, &p2));
            }
            4 => {
                break;
            }
            _ => {
                println!("Wrong instruction, try again")
            }
        };
    }
}
```

Part 2: Applying Concurrency with Rayon

Question 6

The output shows as follows:

```
The average age of people older than 30 is 36.5
```

Question 7

We make some changes in `fn main()` like this:

```
fn main() {
    let v: Vec<Person> = vec![Person { age: 23 },
                              Person { age: 19 },
                              Person { age: 42 },
                              Person { age: 17 },
                              Person { age: 17 },
                              Person { age: 31 },
                              Person { age: 30 }, ];
    let num_over_30 = v.par_iter().filter(|&x| x.age > 30).count() as
f32;
    let sum_over_30: u32 = v.par_iter().map(|x| x.age).filter(|&x| x >
30).sum();
    let avg_over_30 = sum_over_30 as f32 / num_over_30;
    println!("The average age of people older than 30 is {}",
avg_over_30);
}
```

Question 8

```
Original                time:   [17.309 us 17.498 us 17.718 us]

Rayon                   time:   [86.458 us 87.036 us 87.675 us]
```

As we can see, the result with `Rayon` is a little bit slower than the original one.

Question 9

With size `1000` (One-thousand):

```
Original                time:   [2.4933 us 2.5375 us 2.5892 us]

Rayon                   time:   [39.540 us 40.022 us 40.634 us]
```

With size `10000` (Ten-thousand): refer to **Question 8**

With size `100000` (One-hundred-thousand):

```
Original                time:   [183.10 us 187.84 us 193.63 us]
Rayon                   time:   [477.66 us 479.84 us 482.43 us]
```

With size `1000000` (One-million):

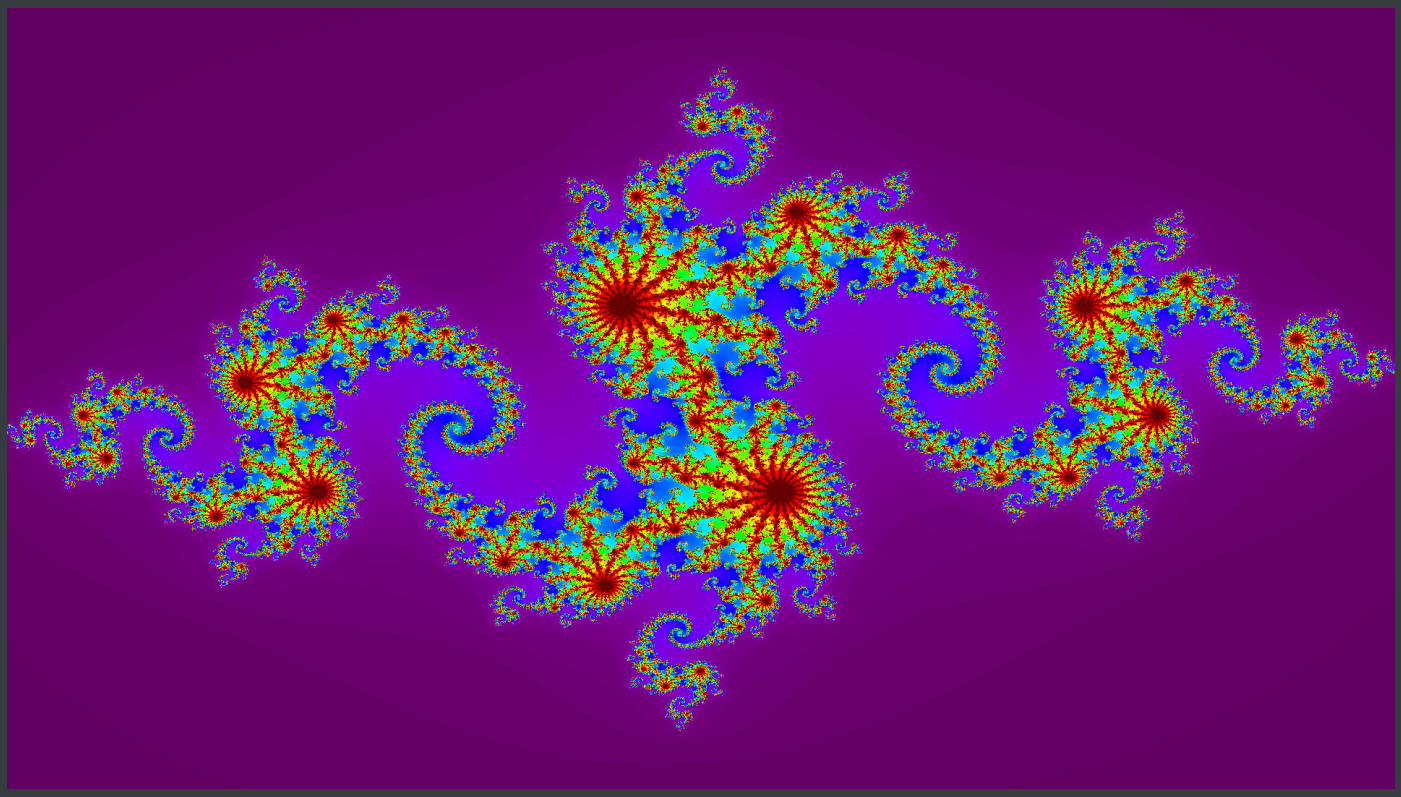
```
Original                time:   [1.6835 ms 1.6977 ms 1.7129 ms]

Rayon                   time:   [3.4808 ms 3.5008 ms 3.5227 ms]
```

From the results we can see that the results using `Rayon` are slower than the traditional solution for all four test examples.

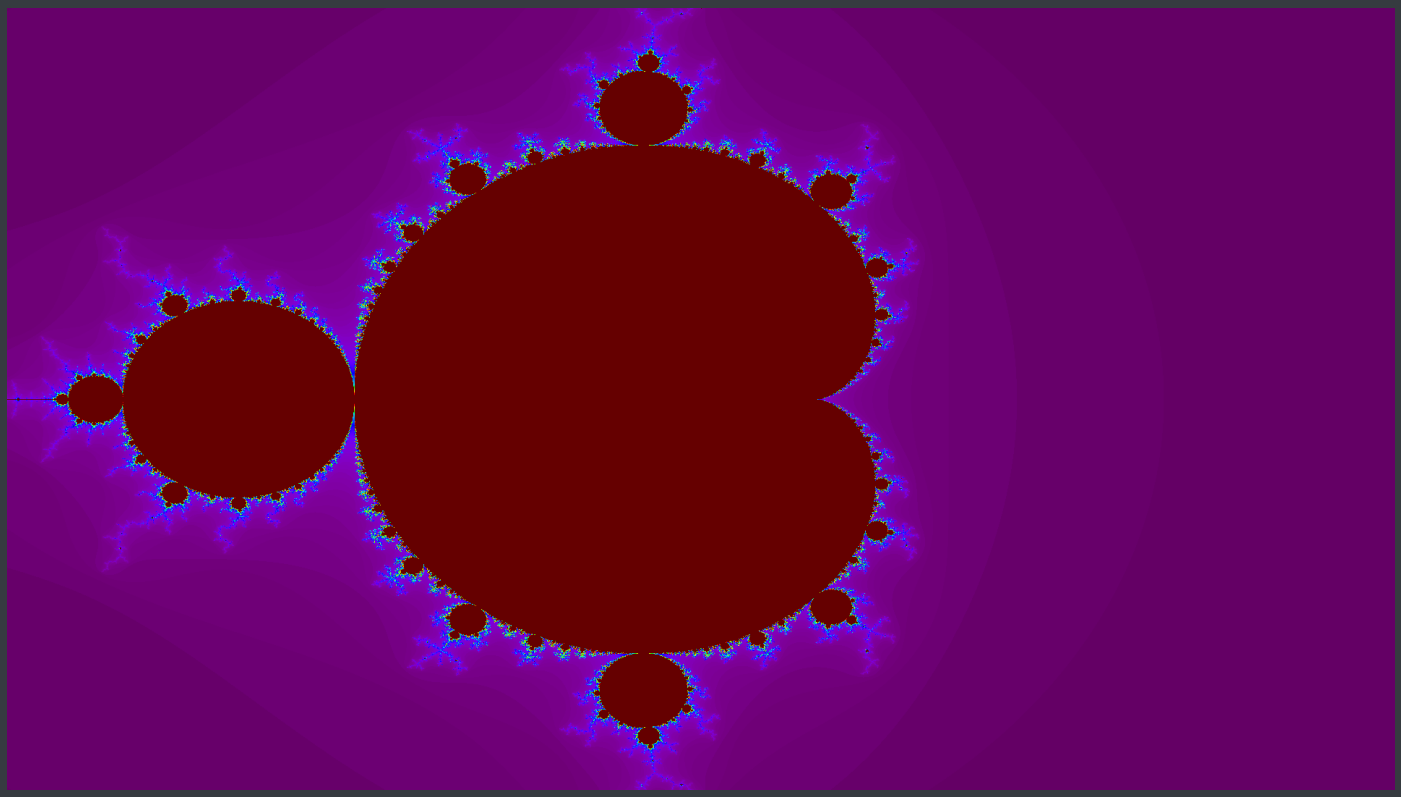
In my view, this is because our operation `let num_over_30 = v.iter().filter(|&x| x.age > 30).count() as f32;` is very simple and easy to do, and, the overhead of creating and managing multiple threads is also very high.

Part 3: Drawing Pixels Concurrently

Before starting Question 10, the sample code will generate a picture like this:



Question 10



We define the *Mandelbrot* function like this:

```
fn mandelbrot(x: u32, y: u32, width: u32, height: u32, max_iter: u32) ->
u32 {
```

```
        let width = width as f32;
        let height = height as f32;

        let mut c = Complex {
            // scale and translate the point to image coordinates
            re: 3.0 * (x as f32 - 0.5 * width) / width,
            im: 2.0 * (y as f32 - 0.5 * height) / height,
        };

        let mut z = Complex {
            // scale and translate the point to image coordinates
            re: 0.0,
            im: 0.0,
        };
        let mut i = 0;
        for t in 0..max_iter {
            if z.norm() >= 2.0 {
                break;
            }
            z = z * z + c;
            i = t;
        }
        i
    }
```

Question 11

We will make the following changes to apply `Rayon` crate:

```
use rayon::prelude::*;

fn main() -> Result<()> {
// ....
    let pool =
rayon::ThreadPoolBuilder::new().num_threads(num_cpus::get()).build().unw
rap();
// ....
    for y in 0..height {
        let tx = tx.clone();
        pool.install(
          // ....
        );
    }
}
```

Question 12

Refer to **Question 11**.