Mandelbrot Area Challenge

Group 5 (7 MEMBERS)

Organization among the Team

Super Villain : Yazeed
Minions: Andrea, Arpan, Felix, Lisa, Mirion, Tristan

ERUM
DATA HUB

Phase 1 Preparation

- Reading and understanding the code
- Defining the problem
- Collecting ideas

Phase 2 Organization

- Who feels comfortable with which tools?
- Split the tasks and ideas accordingly

Phase 3 Work Phase

- Work on the tasks
- Help each other
- Getting everyone involved

Phase 4 World Domination

- Discuss the results
- Decide for a method
- Combine the ideas to get the end result
- Prepare the presentation

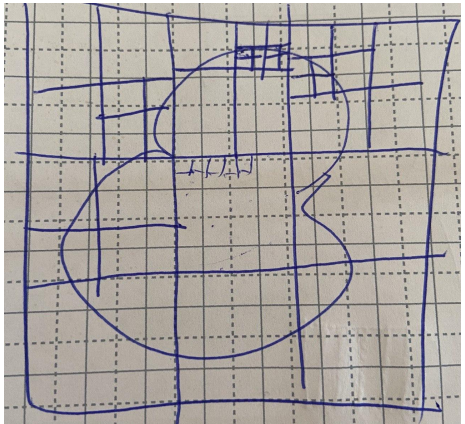
Ansatz

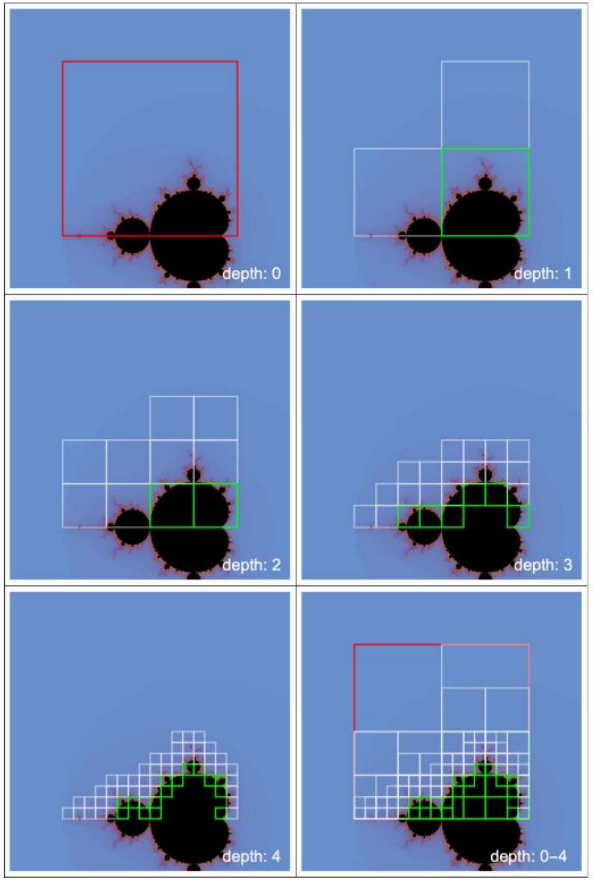
ERUM
DATA HUB

On the Topic of Tessellation

Quad Tessellation: cool but confidence level unclear

Mesh adaption: Works, we could calculate uncertainty, but probably other methods are faster



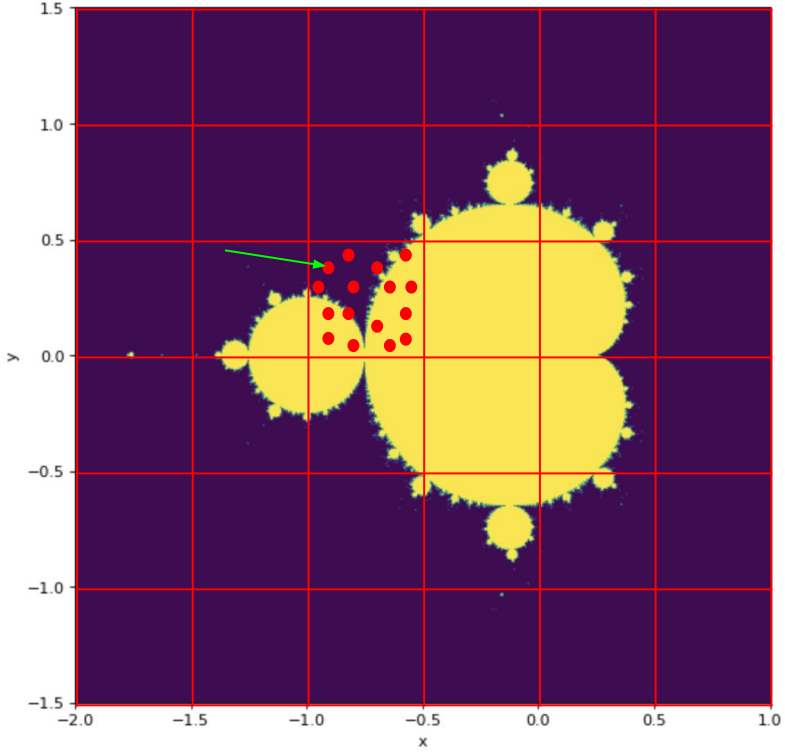


On the Topic of Tessellation

Implement original method.

-> parallelizable

Global Confidence level can be calculated



Know your code

- Identify the bottlenecks in your code
  - 2% CODE TAKES 98% OF TIME !

- Use Jax/numba.cuda for this specific function may improve the most

```python
def is_in_mandelbrot(x, y):
    """Toirtoise and Hare approach to check if point (x,y) is in Mandelbrot set."""
    c = np.complex64(x) + np.complex64(y) * np.complex64(1j)
    z_hare = z_tortoise = np.complex64(0)  # tortoise and hare start at same point
    while True:
        z_hare = z_hare * z_hare + c
        z_hare = (
            z_hare * z_hare + c
        )  # hare does one step more to get ahead of the tortoise
        z_tortoise = z_tortoise * z_tortoise + c  # tortoise is one step behind
        if z_hare == z_tortoise:
            return True  # orbiting or converging to zero
        if z_hare.real**2 + z_hare.imag**2 > 4:
            return False  # diverging to infinity
```

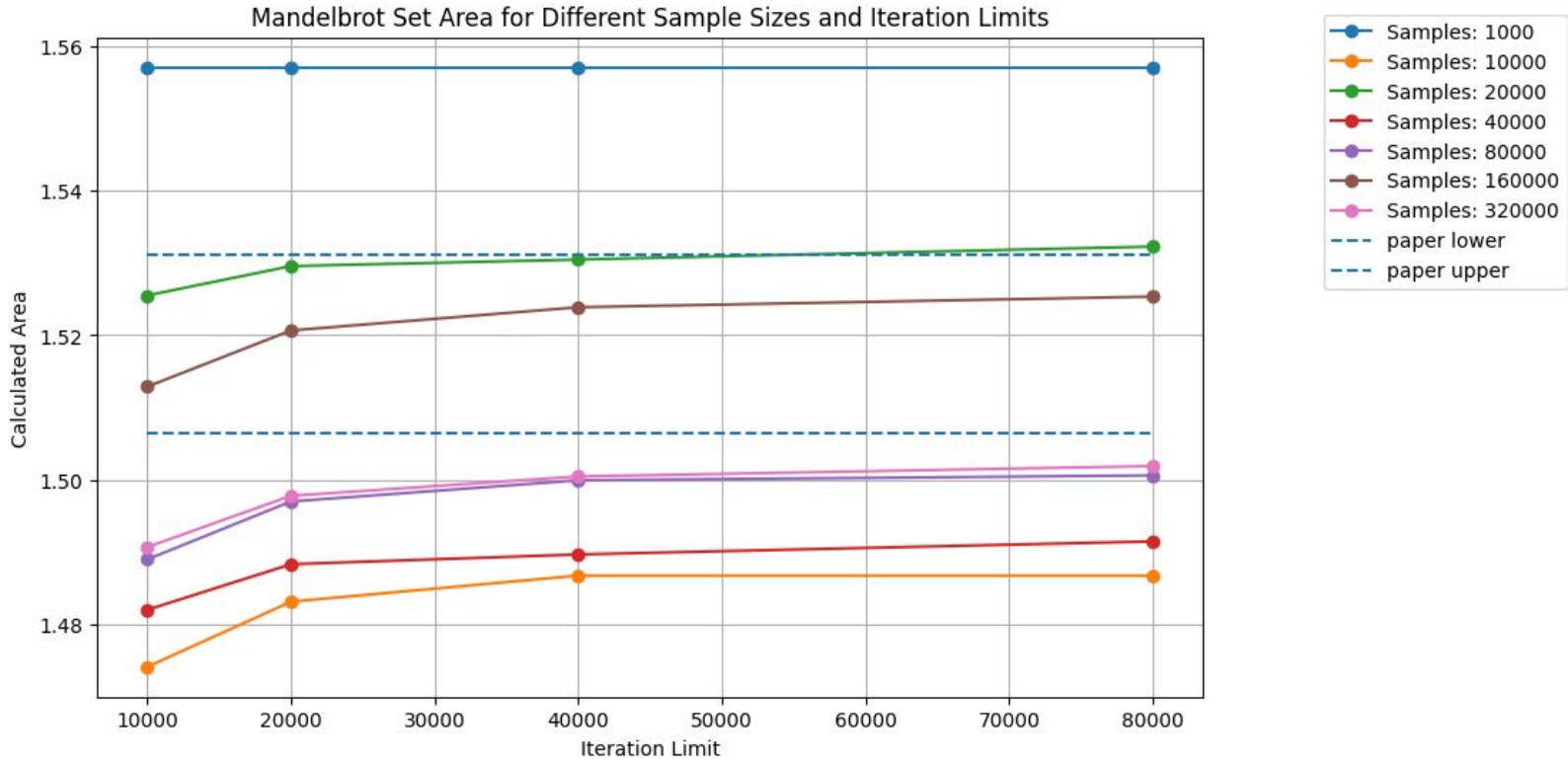
Know your code

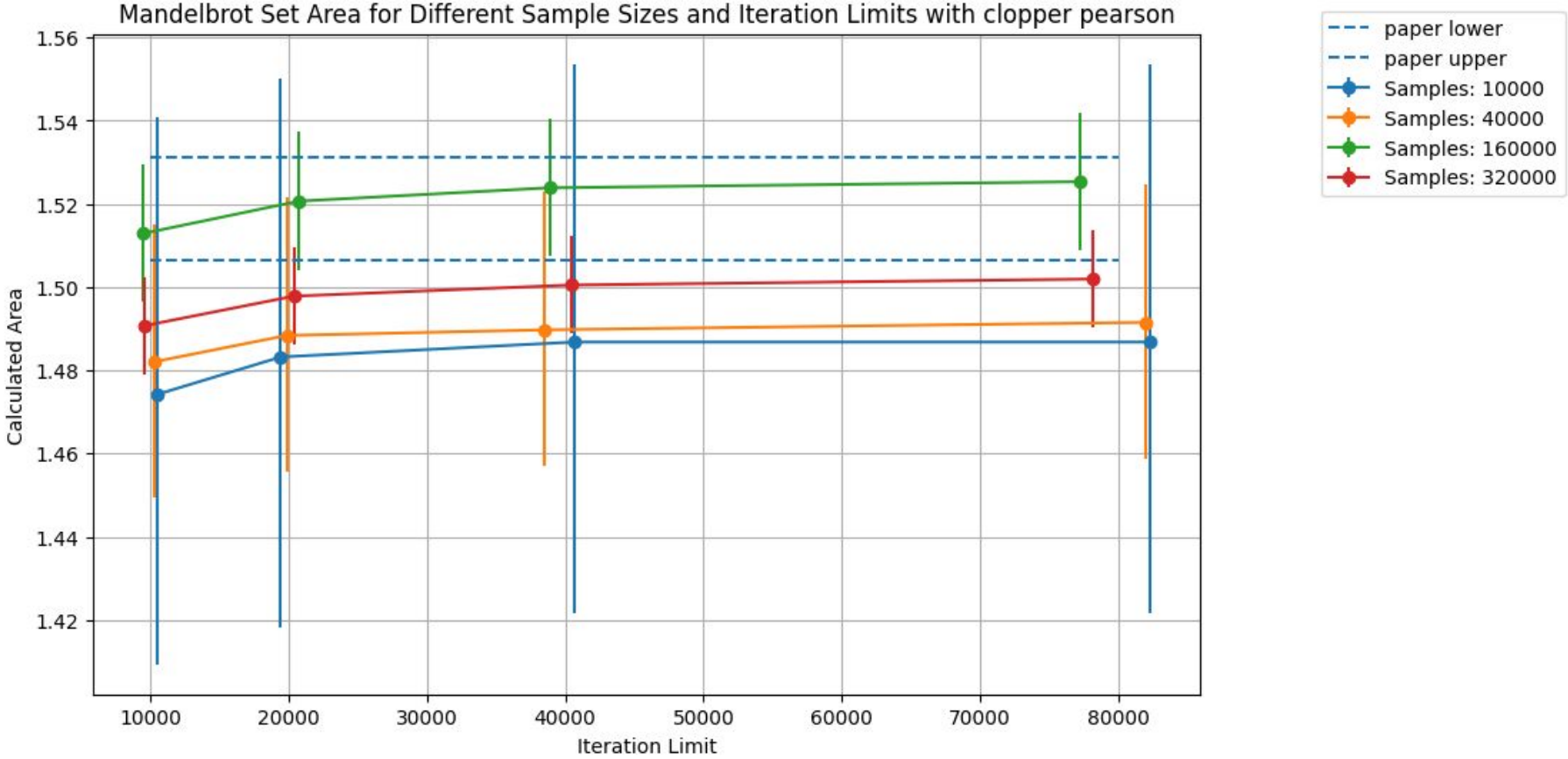
- Identify the bottlenecks in your code
  - 2% CODE TAKES 98% OF TIME !

- Use Jax/numba.cuda for this specific function may improve the most

```
@jax.jit
def is_in_mandelbrot(x, y):
    c = jnp.complex64(x) + jnp.complex64(y) * jnp.complex64(1j)
    z_hare = z_tortoise = jnp.complex64(0)

    def mandelbrot_condition(val):
        z_hare, z_tortoise, iteration, in_set = val
        return jnp.logical_and(iteration < 10000, in_set == -1)

    def mandelbrot_step(val):
        z_hare, z_tortoise, iteration, in_set = val
        z_hare = z_hare * z_hare + c
        z_hare = z_hare * z_hare + c
        z_tortoise = z_tortoise * z_tortoise + c

        diverged = jnp.abs(z_hare.real)**2 + jnp.abs(z_hare.imag)**2 > 4
        collided = jnp.all(z_hare == z_tortoise)

        in_set = jnp.where(collided, 1, jnp.where(diverged, 0, in_set))

        return z_hare, z_tortoise, iteration + 1, in_set

    initial_val = (z_hare, z_tortoise, 0, jnp.array(-1))

    _, _, _, in_set = lax.while_loop(mandelbrot_condition, mandelbrot_step, initial_val)

    return in_set
```

Results

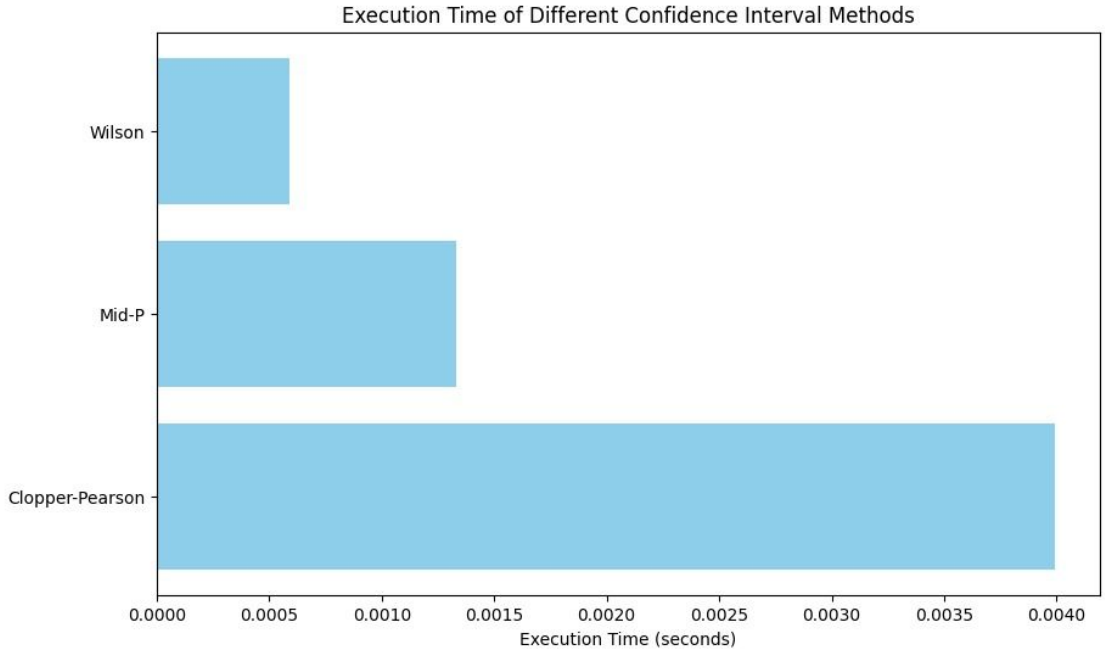

Mandelbrot Set Area for Different Sample Sizes and Iteration Limits
Calculated Area
Iteration Limit
1.56
1.54
1.52
1.50
1.48
10000
20000
30000
40000
50000
60000
70000
80000
Samples: 1000
Samples: 10000
Samples: 20000
Samples: 40000
Samples: 80000
Samples: 160000
Samples: 320000
paper lower
paper upper


Mandelbrot Set Area for Different Sample Sizes and Iteration Limits with clopper pearson
Calculated Area
Iteration Limit
1.56
1.54
1.52
1.50
1.48
1.46
1.44
1.42
10000
20000
30000
40000
50000
60000
70000
80000
paper lower
paper upper
Samples: 10000
Samples: 40000
Samples: 160000
Samples: 320000

ERUM
DATA HUB

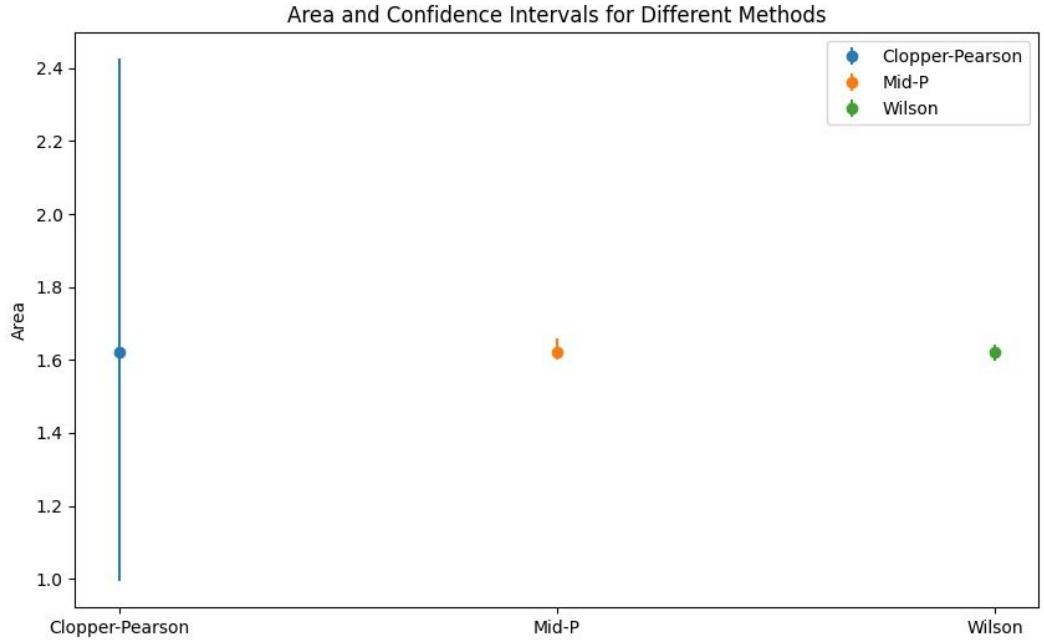
Explore…

- different confidence interval



Explore…

- area for different confidence interval



Explore…

- different uncertainty functions

~7.2s

For Area of 1.5209100000000002: (num samples = 1e5, num itersion 1e6)

0 —> Bootstrap —> 0.3168255671501276 (Num of samples = 1e4)
1 —> Bayesian —> 0.345255
2 —> Monte Carlo —> 0.34183298553533426 (Num of simulation = 1e4)

Using GPU T4 (Colab)

~ few s

Taking full set (with 1M samples, iterations at 100,000):
Area: 1.5047190000000001
Standard Error of the Area: 0.0033583168002794196
Clopper-Pearson Confidence Interval: (1.498141001714686, 1.51131435250625)
Mid-P Confidence Interval: (1.5045086648026536, 1.5049313479427608)
Wilson Confidence Interval: (1.504508422552091, 1.5049296010036375)

~10-15 min

Taking full set (with 1M samples, iterations at 1,000,000):
Area: 1.505601
Standard Error of the Area: 0.0033591032477134426
Clopper-Pearson Confidence Interval: (1.4990214577538137, 1.5121978913533467)
Mid-P Confidence Interval: (1.5053906147498148, 1.5058133974030046)
Wilson Confidence Interval: (1.5053903732330385, 1.5058116503157537)

Conclusions & Outlook

Conclusion

First: Know your code → Try to optimize the 2% of the code that takes the most time.

Second: breadth-first search

Third: Where do you need the GPU?