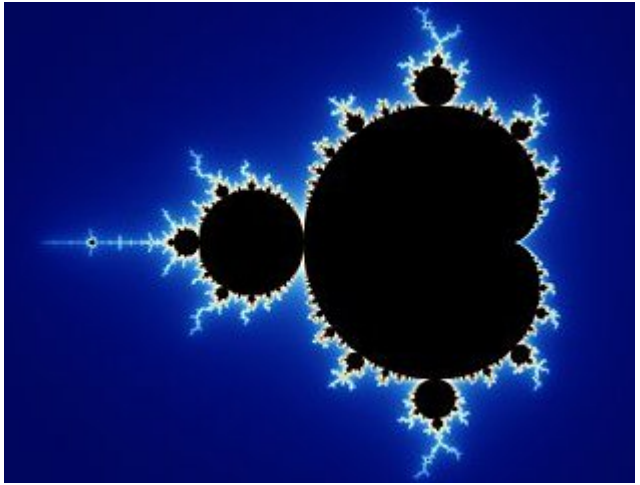
Mandelbrot Area Challenge

Green group:

Lukas Beiske, Keerthana Chand, Alec Hilberg, Leandro Intelisano, Felix Krückel, Tereza Vaclavu, Felix Zinn

Organization among the Team

- All team members get into the task
- Discussion about the possibilities and which direction we should go
- Split the team in 3 subgroups
  - One subgroup tries to implement the functionality with JAX
  - The one subgroup also try rewrite code to Numba-Cuda
  - Other subgroup tries to find structural optimizations
    - We found the mirror axis of the image – tries to improve the code by introducing geometrical restrictions



source: wikipedia

Ansatz

ERUM DATA HUB

Plan A: JAX

- try to rewrite everything in JAX

- test if this this approach speeds up the calculation

Plan B: Numba Cuda

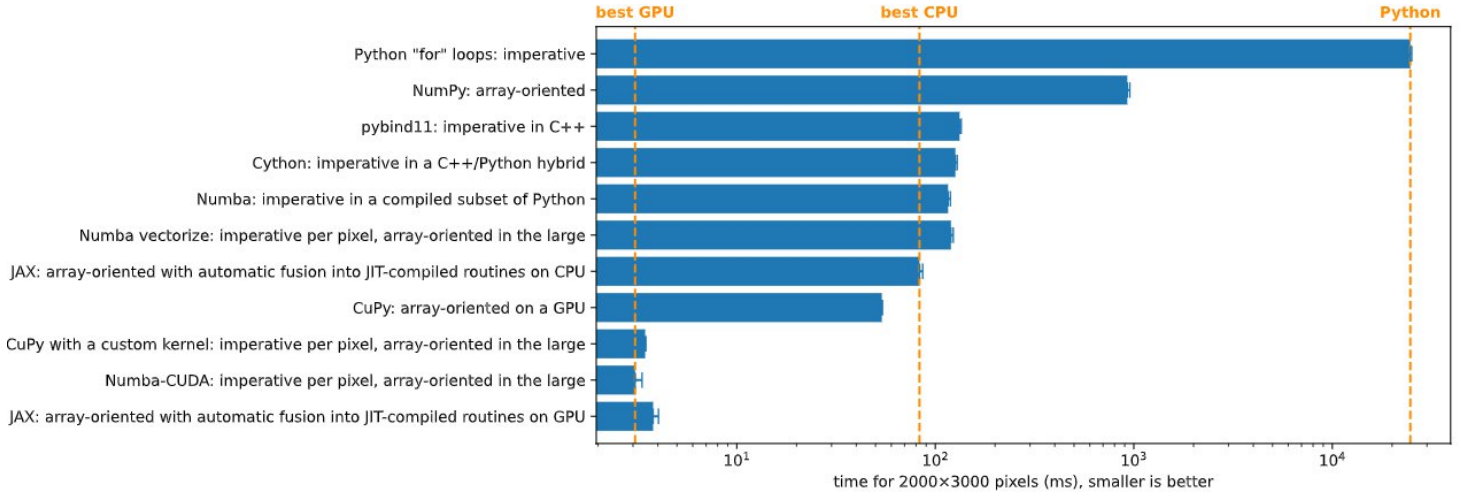
- the plan with JAX not working properly, the next step try in Numba Cuda

- rewrite part of the code in cuda to speed up

Improvements

- try to improve the code only by using the mirror axis (in y direction) of the image

– implement in used code



credits: https://gist.github.com/jpivarski/da343abd8024834ee8c5aaba691aafc7#file-plot-mandelbrot-on-all-accelerators-colab-svg

Results

ERUM DATA HUB

Plan A: JAX

Yesterday:

```
terminated  python challenge_jax.py
```

Today:
15 tiles: 1.5080 ($4.1e^{-3}$)

→ while loops in jax are difficult... and if statements, too...

Plan B: Numba Cuda

Converted the count_mandelbrot function.

Didn't have the time to finish the full pipeline.

The idea was to calculate different tiles using Streams and save .npy files of the numers.

Improvements: Geometry

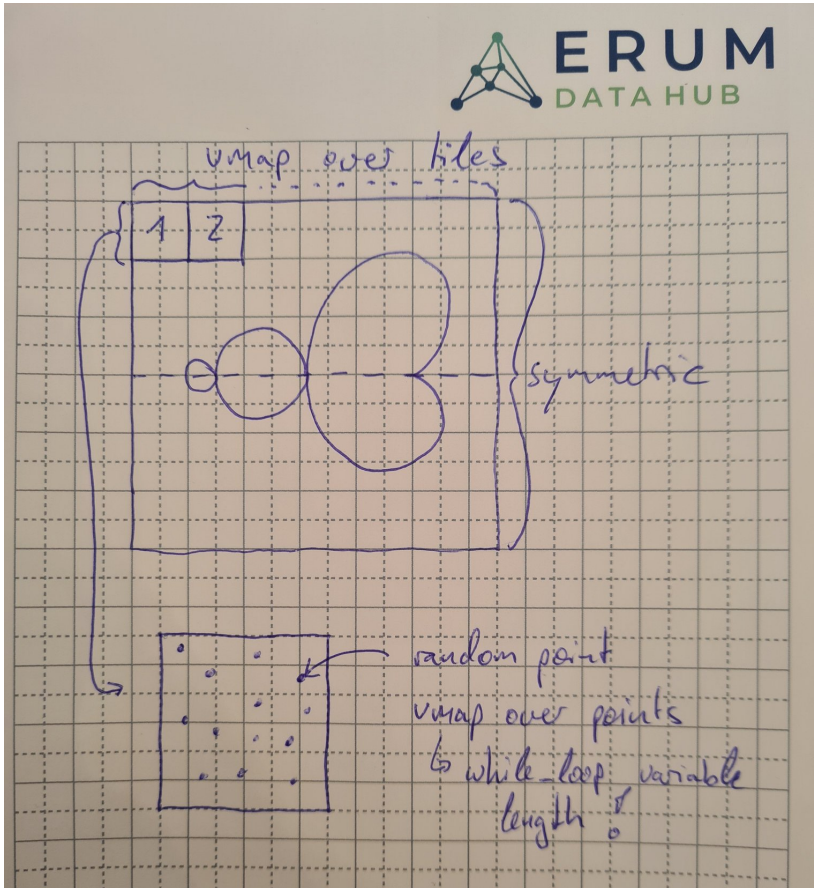
100 tiles, samples in batch 100

Used code:
1.50657 ($1.39e^{-4}$) - 18 s

Apply the geometry restrictions:
1.50649 ($1.6e^{-4}$) - 13 s

Speed up proved
> Check with GPU solutions necessary

JAX Strategy



Difficulties

```python
def is_in_mandelbrot(x, y):
    """Toirtoise and Hare approach to check if point (x,y) is in Mandelbrot set."""
    c = jnp.complex64(x) + jnp.complex64(y) * jnp.complex64(1j)
    z_hare = z_tortoise = jnp.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
```

```python
@jax.jit
def mandelbrot_jax(x, y):
    c = x + y * 1j
    init_vals = (jnp.complex64(0), jnp.complex64(0), c)

    def cond_fun(val):
        diverge = (val[0].real ** 2 + val[0].imag ** 2) > 4
        not_converge = val[0] != val[1]
        cond = ~diverge & not_converge
        return cond

    def body_fun(val):
        zhare, ztort, c = val

        zhare = zhare * zhare + c
        zhare = zhare * zhare + c

        ztort = ztort * ztort + c

        return (zhare, ztort, c)

    return jax.lax.while_loop(cond_fun, body_fun, body_fun(init_vals))
```

Outlook

- Further debugging of JAX solution to get a fully working and speed-wise improved solution
- Finishing the numba-cuda approach to have another comparison for the uncertainty and run-time
- Implementation of geometrical restriction in all GPU approaches to further improve run-time