Mandelbrot Area Challenge

GROUP 7:
Max Fuste Costa, Ahmar Khaliq, Oleksandr Koshchii, Arul Parkash, Jannis Schaeper, Sebastian Vetter, Sarah Wagner

Organization among the Team

Organization among the Team

- Optimization of the RNG spawner
- Profiling
- Trying to port the numba JIT to jax
- Vectorizing the most often used functions

Ansatz

Ansatz

- go through provided code “challenge.py” and “challenge.ipynb”
    - running this on CPU only yields an area of 1.50687 +/- 0.00014 in 15s of runtime
- find bottlenecks with profiling and improve expensive parts
- lots of different approaches for improving the bottlenecks
- compare: a) initial number tiles and time it takes on CPU without code modification, b) implement GPU calculation with improvements and with CUDA

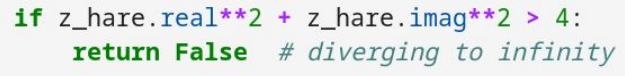
Improvements

Improvements

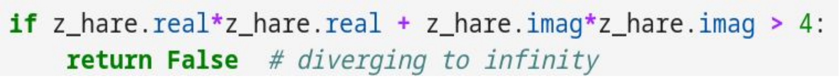
- Mandelbrot is symmetric on imaginary axis —> considering half is enough
- types: np.complex slower than python complex
    - `c = np.complex64(x) + np.complex64(y) * np.complex64(1j)   —> c = x + y * 1j`
-

Improvements

- Results of Profiling on *"is_in_mandelbrot(x, y)"* :
    - Replacing square function by multiplication reduces if's time consumption from 74.8% to 63.1% of loop time consumption

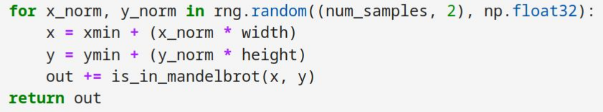


```
if z_hare.real**2 + z_hare.imag**2 > 4:
    return False  # diverging to infinity
```

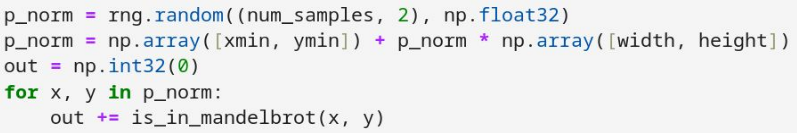


```
if z_hare.real*z_hare.real + z_hare.imag*z_hare.imag > 4:
    return False  # diverging to infinity
```

-
    - scaling outside of for loop:



```
for x_norm, y_norm in rng.random((num_samples, 2), np.float32):
    x = xmin + (x_norm * width)
    y = ymin + (y_norm * height)
    out += is_in_mandelbrot(x, y)
return out
```



```
p_norm = rng.random((num_samples, 2), np.float32)
p_norm = np.array([xmin, ymin]) + p_norm * np.array([width, height])
out = np.int32(0)
for x, y in p_norm:
    out += is_in_mandelbrot(x, y)
```

Results

Result

Area of Mandebrot set:
1.5065985375000004 +/- 1.6316288545394807e-05
Calculated in 8s

Outlook

Outlook

- One big problem for parallelization: while-loop in the *is_in_mandelbrot* computation
    - Calculated a fixed number of iterations (enough that xx% converge or diverge) and calculate the if-else branching vectorized on this
    - Make function output two boolean arrays instead of returning when condition is met
- Correctly implement the CUDA-kernel in numba (or even directly in CUDA)