Zbiór Julii za pomocą procesora CPU:

def julia(x, y, max\_iters):

  i = 0

  c = complex(-0.8, 0.156)

  a = complex(x,y)

  for i in range(max\_iters):

    a = a\*a + c

    if (a.real\*a.real + a.imag\*a.imag) > 1000:

      return int((i/max\_iters)\*255)

  return 255

def create\_fractal(min\_x, max\_x, min\_y, max\_y, image, iters):

  height = image.shape[0]

  width = image.shape[1]

  pixel\_size\_x = (max\_x - min\_x) / width

  pixel\_size\_y = (max\_y - min\_y) / height

  for x in range(width):

    real = min\_x + x \* pixel\_size\_x

    for y in range(height):

      imag = min\_y + y \* pixel\_size\_y

      color = julia(real, imag, iters)

      image[y, x] = color

  return image

import numpy as np

import matplotlib.pyplot as plt

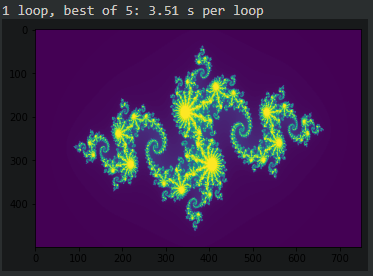
image = np.zeros((500, 750), dtype=np.uint8)

%timeit create\_fractal(-2.0, 2.0, -1.0, 1.0, image, 200)

plt.imshow(image)

plt.viridis()

plt.show()



Zbiór Julii za pomocą procesora GPU:

@cuda.jit(device=True)

def julia(x, y, max\_iters):

  i = 0

  c = complex(-0.8, 0.156)

  a = complex(x,y)

  for i in range(max\_iters):

    a = a\*a + c

    if (a.real\*a.real + a.imag\*a.imag) > 1000:

      return int((i/max\_iters)\*255)

  return 255

threadsperblock = 16

xblocks = (image.shape[1] + (threadsperblock - 1)) // threadsperblock

yblocks = (image.shape[0] + (threadsperblock - 1)) // threadsperblock

@cuda.jit

def create\_fractal(min\_x, max\_x, min\_y, max\_y, image, iters):

  height = image.shape[0]

  width = image.shape[1]

  pixel\_size\_x = (max\_x - min\_x) / width

  pixel\_size\_y = (max\_y - min\_y) / height

  x, y = cuda.grid(2)

  if x < width and y < height:

    real = min\_x + x \* pixel\_size\_x

    imag = min\_y + y \* pixel\_size\_y

    color = julia(real, imag, iters)

    image[y, x] = color

image = np.zeros((1000, 1000), dtype=np.uint8)

threadsperblock = 16

threads\_per\_block = (16, 16)

xblocks = (image.shape[1] + (threadsperblock - 1)) // threadsperblock

yblocks = (image.shape[0] + (threadsperblock - 1)) // threadsperblock

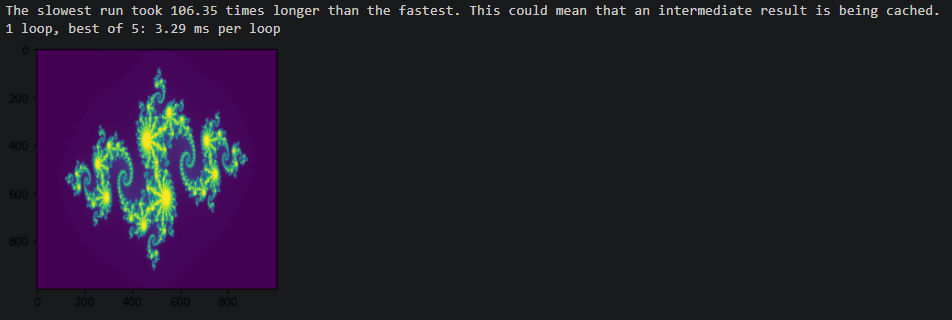
blocks\_per\_grid = (xblocks, yblocks)

%timeit create\_fractal[blocks\_per\_grid, threads\_per\_block](-2.0, 2.0, -1.0, 1.0, image, 200)

plt.imshow(image)

plt.viridis()

plt.show()



Sumowanie wektorów za pomocą procesora CPU:

%%cu

#define N 100000

#include <stdio.h>

#include <time.h>

void add( int \*a, int \*b, int \*c ) {

int tid = 0; // to jest CPU nr zero, a więc zaczynamy od zera

while (tid < N) {

c[tid] = a[tid] \* b[tid];

tid += 1; // mamy tylko jeden CPU, a więc zwiększamy o jeden

  }

}

int main( void ) {

int a[N], b[N], c[N];

clock\_t start, end;

// zapelnianie tablic a i b za pomocą CPU

for (int i=0; i<N; i++) {

a[i] = -i;

b[i] = i \* i;

}

start = clock();

add( a, b, c );

end = clock();

double CPU = ((double)(end-start))/CLOCKS\_PER\_SEC;

printf("CPU: %lf sekund \n", CPU);

return 0;

}

CPU: 0.000496 sekund

Sumowanie wektorów za pomocą procesora GPU:

%%cu

#define N 100000

#include <stdio.h>

#include <time.h>

\_\_global\_\_ void add( int \*a, int \*b, int \*c ) {

int tid = blockIdx.x; // dodawanie na danych znajdujących się pod indeksem

if (tid < N)

c[tid] = a[tid] \* b[tid];

}

int main( void ) {

int a[N], b[N], c[N];

int \*dev\_a, \*dev\_b, \*dev\_c;

// alokacja pamięci na GPU

cudaMalloc( (void\*\*)&dev\_a, N \* sizeof(int) );

cudaMalloc( (void\*\*)&dev\_b, N \* sizeof(int) );

cudaMalloc( (void\*\*)&dev\_c, N \* sizeof(int) );

// zapelnienie tablic a i b na CPU

for (int i=0; i<N; i++) {

a[i] = -i;

b[i] = i \* i;

}

// kopiowanie tablic a i b do GPU

clock\_t start, end;

start = clock();

cudaMemcpy( dev\_a, a, N \* sizeof(int), cudaMemcpyHostToDevice );

cudaMemcpy( dev\_b, b, N \* sizeof(int), cudaMemcpyHostToDevice );

add<<<N,1>>>( dev\_a, dev\_b, dev\_c );

// kopiowanie tablicy c z GPU do CPU

cudaMemcpy( c, dev\_c, N \* sizeof(int), cudaMemcpyDeviceToHost );

// zwolnienie pamieci alokowanej na CPU

cudaFree( dev\_a );

cudaFree( dev\_b );

cudaFree( dev\_c );

end = clock();

double GPU = ((double)(end-start))/CLOCKS\_PER\_SEC;

printf("GPU: %lf sekund \n", GPU);

return 0;

}

GPU: 0.001527 sekund

Podsumowanie:

Wykonanie fraktalu zbioru Julii na CPU wynosi aż 3.51s, gdy na GPU wykonywane jest to w ciągu 3.29ms co daje nam znaczącą różnicę w czasu dzięki temu że GPU wykorzystuje o wiele więcej wątków, CPU nie nadrabia tego swoją wyższą częstotliwością taktowania.

Sumowanie wektorów za pomocą procesora CPU wynosi tylko 0.000496s z powodu małej wielkości tablicy a CPU radzi sobie lepiej z mniejszymi problemami niż bardziej rozległymi, zaś za pomocą GPU, 0.001527