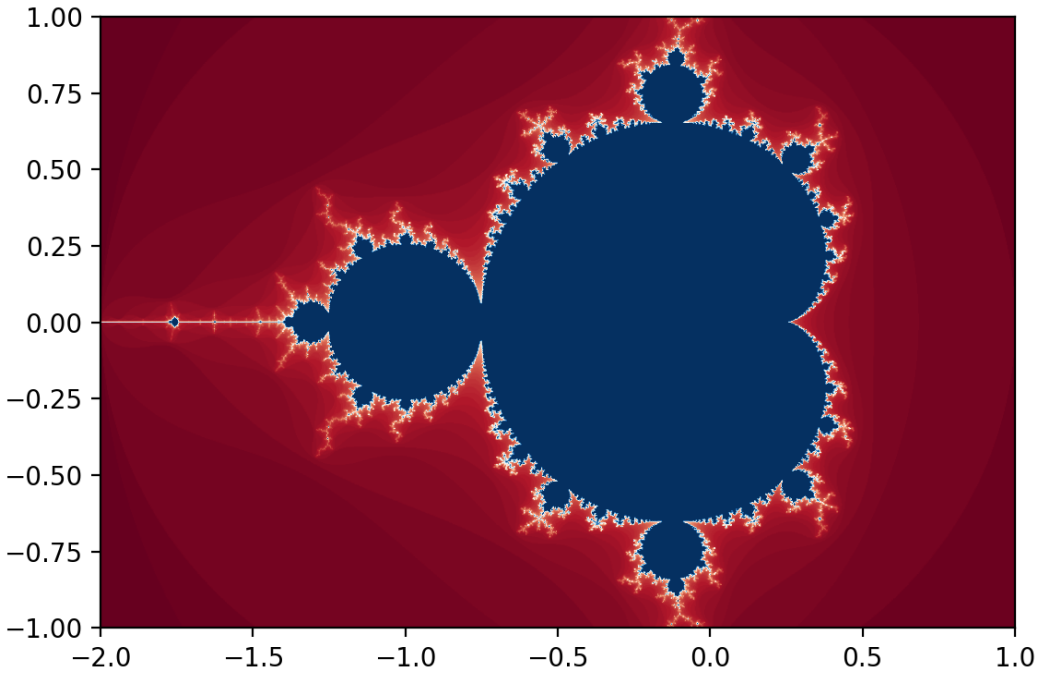


Plotting Mandelbrot Set in Python

The Math:

$z_0 = c$

$z_{n+1} = {z_n}^2 + c$ where $z_k \in \mathbb{C}$

The points z_{n+1} that blows infinity become a part of Mandelbrot set.

GPU Accelerated Python Code

`from numba import cuda`

This imports a GPU computation wrapper from a speed optimization library called `numba`.

`import numpy as np`

This imports numeric computation library as np. Python doesn't have arrays. This allows us to use arrays.

`import matplotlib.pyplot as plt`

This line imports a plotting library. Since it's name `matplotlib.pyplot` is too long. We will use the library by using variable `plt`

```
def mapFromTo(x, a, b, c, d):
        y = (x - a) / (b - a) * (d - c) + c
        return y
```

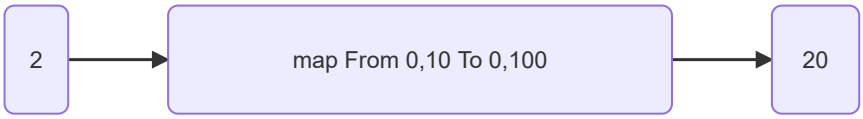
`^ Courtesy of StackOverflow`

This functions maps input from one range to another.

This will map a value x from range (a, b) to (d, c).

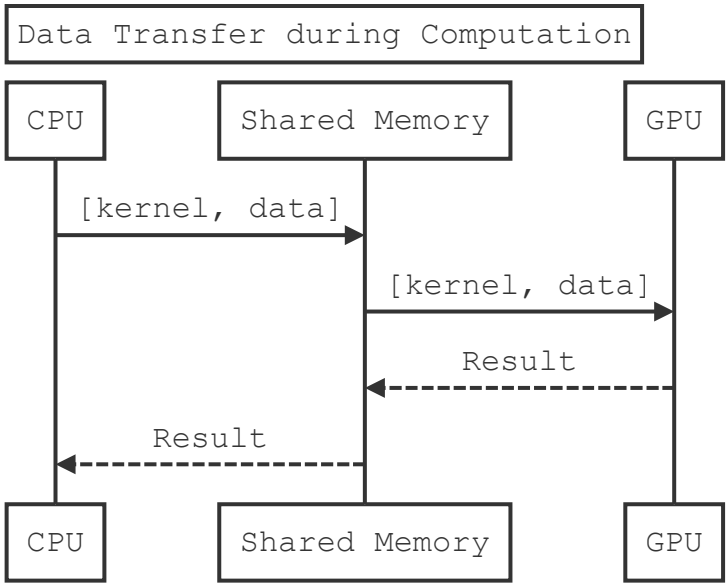
Example:

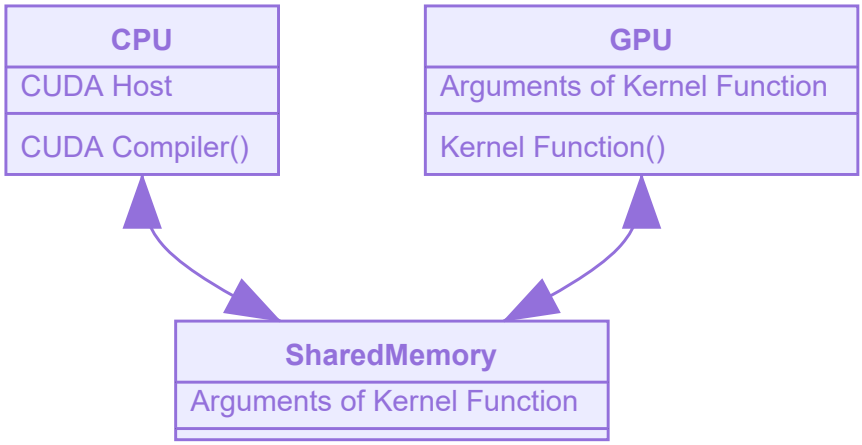
2 when mapped from (0, 10) to (0, 100) becomes 20.



GPU Computation Model:

The functions used in GPU computation shouldn't return an object. Instead I/O objects should be passed as an argument to the function. GPU has it's own memory and processing units therefore returns are meaningless use since they won't return anything to the host computer. GPUs perform operations on objects in shared memory. The I/O arrays are stored in shared memory hence accessible to both host and GPU.





```
@cuda.jit
def kernel(input_array, output_array):
    do_something
```

The `kernel` function is an ordinary python function. The

`@cuda.jit` wrapper compiles the function for GPU.

Mandelbrot Kernel:

```
def mandelbrot_kernel(data, xlow, xhigh, ylow, yhigh):
    tx = cuda.threadIdx.x
    ty = cuda.blockIdx.y
    bw = cuda.blockDim.x

    x = mapFromTo(tx, 0, row, xlow, xhigh)
    y = mapFromTo(ty, 0, col, ylow, yhigh)
    c = complex(x, y)
    z = 0.0j
    max_iters = 50
    for i in range(max_iters):
        z = z ** 2 + c
        if (z.real ** 2 + z.imag ** 2) >= 4:
            data[tx, ty] = i
            break
        else:
            data[tx, ty] = 100
```

Block 0	Threads X	Thread Y
	0	0
	1	1
	...	...
	1000	1000

Full Code

```
from numba import cuda
import numpy as np
import matplotlib.pyplot as plt


@cuda.jit
def mandelbrot_kernel(data, xlow, xhigh, ylow, yhigh):
    tx = cuda.threadIdx.x
    ty = cuda.blockIdx.x
    bw = cuda.blockDim.x
```

```
    def mapFromTo(x, a, b, c, d):
        y = (x - a) / (b - a) * (d - c) + c
        return y


    x = mapFromTo(tx, 0, row, xlow, xhigh)
    y = mapFromTo(ty, 0, col, ylow, yhigh)
    c = complex(x, y)
    z = 0.0j
    for i in range(50):
        z = z ** 2 + c
        if (z.real ** 2 + z.imag ** 2) >= 4:
            data[tx, ty] = i
            break
        else:
            data[tx, ty] = 100

row = 1000
col = 1000
plot = np.zeros([row, col])
mandelbrot_kernel[row, col](plot, -2, 1, -1, 1)

fig = plt.figure(dpi=200)
ax = fig.add_subplot(1, 1, 1)
im = ax.imshow(plot.T, cmap="RdBu", interpolation="bilinear")

plt.show()
```