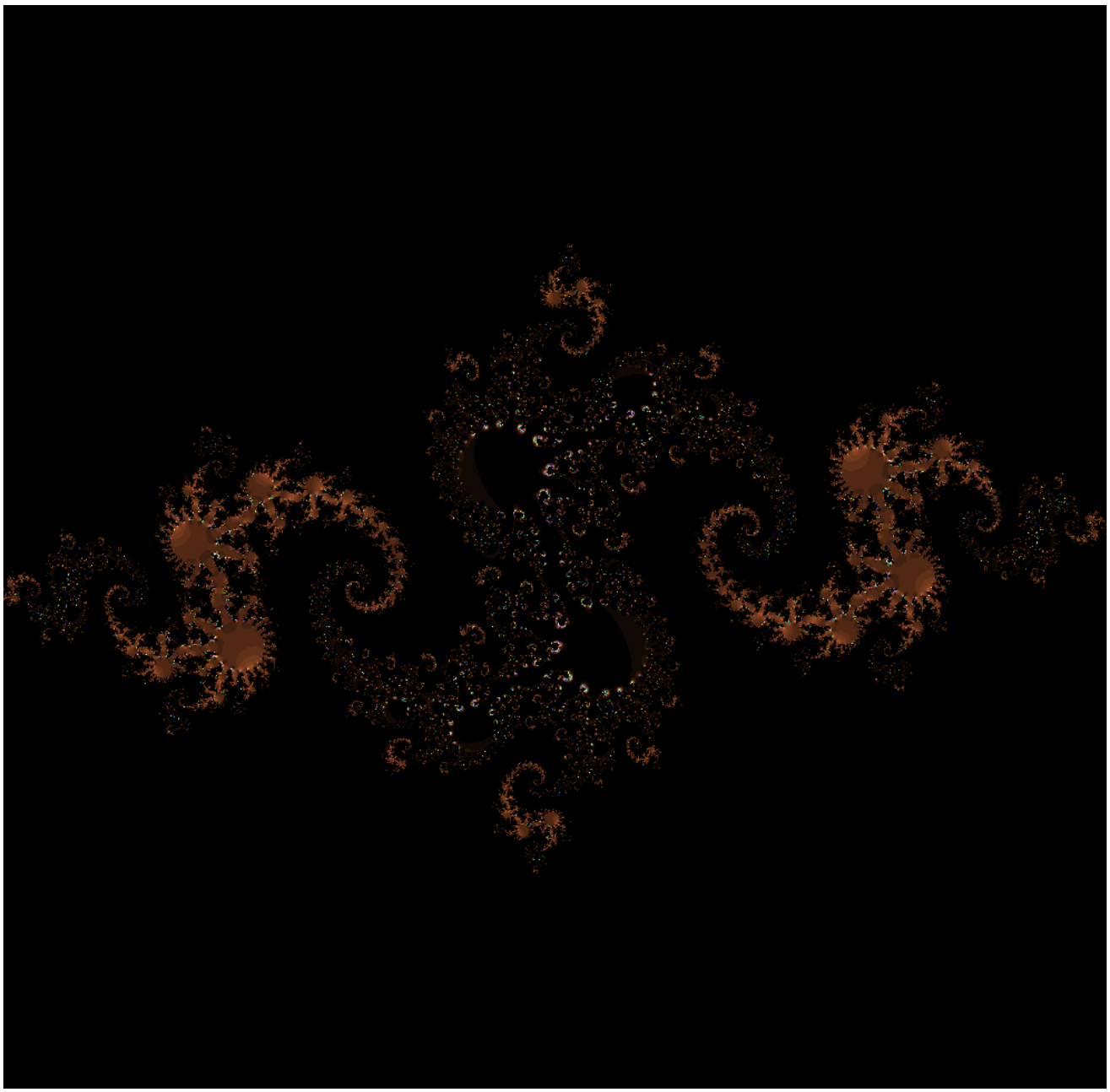
GPU Computing Assignment 1



Code:

```
#include "cuda_runtime.h"
#include "device_launch_parameters.h"
#include <opencv2/highgui/highgui.hpp>
#include <cstdio>
#include <cmath>

using namespace cv;

int DIM;

const float FCT = 2.85, hFCT = FCT/2.0;
class cuComplex
{
    float r,i;
public:
    __device__ cuComplex(float _r,float _i):r(_r),i(_i){}
    __device__ float square()
    {
        return r*r + i*i;
    }
    __device__ cuComplex operator*(const cuComplex& rhs)
    {
        return cuComplex(r*rhs.r - i*rhs.i, i*rhs.r + r*rhs.i);
    }
    __device__ cuComplex operator+(const cuComplex& rhs)
    {
        return cuComplex(r+rhs.r,i+rhs.i);
    }
};

__device__ int julia(int x,int y)
{
    cuComplex c(-0.8, 0.156);
    const float scale = 1.5;
    float jx = scale * (float)(1000/2 - x) / (1000/2);
    float jy = scale * (float)(1000/2 - y) / (1000/2);

    cuComplex a(jx,jy);

    for(int i=0; i<100; i++)
    {
        a = a*a + c;
        if(a.square()>100000 || a.square() < -100000)
            return 0;
    }
    return int(a.square()*(1<<3))%255;
}

__global__ void kernel(unsigned char *ptr)
{
    int x = blockIdx.x*blockDim.x+threadIdx.x;
    int y = blockIdx.y*blockDim.y+threadIdx.y;

    int offset = x+y*1000;
```

```
    if( x > 1000 || y>1000 || offset > 1000*1000 ) return;
//    int x = threadIdx.x + blockIdx.x*blockDim.x;
//    int y = threadIdx.y + blockIdx.y*blockDim.y;

    int juliaValue = julia(x,y);
//    ptr[offset * 3 + 0] = 0;
//    ptr[offset * 3 + 1] = 0;
//    ptr[offset * 3 + 2] = (juliaValue!=0)*255;
    ptr[offset * 3 + 0] = juliaValue<<2;
    ptr[offset * 3 + 1] = juliaValue<<3;
    ptr[offset * 3 + 2] = juliaValue<<4;
}

struct DataBlock
{
    unsigned  char *dev_bitmap;
};

int main()
{

    DataBlock data;
    cudaError_t err;

    Mat image(1000,1000,CV_8UC3,Scalar::all(0));
    data.dev_bitmap = image.data;
    unsigned char* dev_bitmap;

    err = cudaMalloc((void**)&dev_bitmap, 3*image.cols*image.rows);
    data.dev_bitmap = dev_bitmap;
    for(DIM = 32; DIM<1025; DIM<<=1)
    {

        dim3 grid(DIM,DIM);
        cudaEvent_t start1;
        cudaEventCreate(&start1);
        cudaEvent_t stop1;
        cudaEventCreate(&stop1);
        cudaEventRecord(start1, NULL);
        kernel<<<dim3(DIM,DIM),dim3(1000/DIM+1,1000/DIM+1,1)>>> (dev_bitmap);
        cudaEventRecord(stop1, NULL);
        cudaEventSynchronize(stop1);
        float msecTotal1 = 0.0f;
        cudaEventElapsedTime(&msecTotal1, start1, stop1);

        printf("DIM = %d, time = %f\n",DIM,msecTotal1);
        err = cudaMemcpy(image.data, dev_bitmap,
3*image.cols*image.rows,cudaMemcpyDeviceToHost);



        imshow("CUDA Julia", image);

        waitKey();
```

```
    }
    err = cudaFree(dev_bitmap);


    return 0;
}
```

Time:

```
DIM = 32, time = 1.182752
DIM = 64, time = 1.070176
DIM = 128, time = 1.078432
DIM = 256, time = 1.982976
DIM = 512, time = 7.327264
DIM = 1024, time = 27.204321
```

Platform:

```
 GPU: "GeForce GTX 750"
 CUDA:
  CUDA Driver Version / Runtime Version          8.0 / 8.0
  CUDA Capability Major/Minor version number:    5.0
```