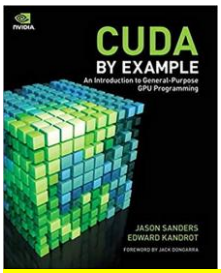
PRiR lab 8
Fraktal Julii z CUDA C

Poniżej zaprezentowano generowanie zbioru Julii na CPU i GPU. Kod źródłowy został oparty na programie z następującej pozycji



Jason Sanders, Edward Kandrot:
CUDA by Example: An Introduction to General-Purpose GPU Programming
***CUDA w przykładach. Wprowadzenie do ogólnego programowania procesorów GPU*. Helion**
https://github.com/CodedK/CUDA-by-Example-source-code-for-the-book-s-examples-

z modyfikacjami dostosowującymi go do Google Colab (dotyczącymi zapisu fraktala do pliku).

Zadanie 2p

1. Dokonaj przekształcenia plików z wygenerowanymi fraktalami do postaci plików graficznych, które wyświetlisz.
2. Przedstaw na wykresach porównanie czasów wykonania programów na GPU (Colab) oraz CPU (Colab oraz komputer stacjonarny) w zależności od rozmiaru problemu (stała DIM). W optymalnym przypadku wykresy powinny zostać zrealizowane z matplotlib i dołączone do Colab.
3. Punkty 1,2 zrealizuj dla fraktala Mandelbrota.
4. Zrealizuj pracę w postaci sprawozdania w Google Colab – dokonaj także opisu programów i zrealizowanych czynności – ma to mieć postać tutoriala dla osoby, która nie na związku z przedmiotem.

Dodatkowe informacje

1. Kod (z opisem) powinien pojawić się na stronie **Google Colab** (proszę podesłać link na **kszerszen.uwb@gmail.com**).
2. Czas zrealizowania zadania – do **18.12.2022**. Każdy tydzień opóźnienia nadesłania kodu obniża punktację o 25% aż do granicy 25%.
3. Do programu powinien zostać dołączony opis z wynikami.
4. Ocena - standardowa implementacja kodu 80% plus wasza innowacyjność 20%.

Wersja A – w C na CPU

```
%%cu
#include <stdio.h>
#define DIM 1000

struct cuComplex{
  float r;
  float i;
  cuComplex(float a,float b):r(a),i(b){}
  float magnitude2(void){return r*r+i*i;}
  cuComplex operator*(const cuComplex &a){
    return cuComplex(r*a.r-i*a.i,i*a.r+r*a.i);
  }
  cuComplex operator+(const cuComplex &a){
    return cuComplex(r+a.r,i+a.i);
  }
};

int julia(int x,int y){
  const float scale=1.5;
  float jx=scale*(float)(DIM/2-x)/(DIM/2);
  float jy=scale*(float)(DIM/2-y)/(DIM/2);
  cuComplex c(-0.8,0.156);
  cuComplex a(jx,jy);
  int i=0;
  for(i=0;i<200;i++){
    a=a*a+c;
    if(a.magnitude2()>1000) return 0;
  }

return 1;
}
```

```
void kernel(unsigned char *ptr){
  for(int y=0;y<DIM;y++){
    for(int x=0;x<DIM;x++){
      int offset=x+y*DIM;
      int juliaValue=julia(x,y);
      ptr[offset*4+0]=255*juliaValue;
      ptr[offset*4+1]=0;
      ptr[offset*4+2]=0;
      ptr[offset*4+3]=255;
    }
  }
}

//Dalsze elementy programu pozwalają za zapisanie Fraktala do pliku tekstowego
struct DataBlock{
  unsigned char *dev_bitmap;
};

void wypisz_RGB_fraktal(unsigned char *ptr){
  for(int y=0;y<DIM;y++){
    for(int x=0;x<DIM;x++){
      int offset=x+y*DIM;

      printf( "\n%d, %d, %d, %d", ptr[offset*4+0], ptr[offset*4+1], ptr[offset*4+2], ptr[offset*4+3]);
    }
  }
}
```

```
void save_to_file(unsigned char *ptr){
    FILE *fp=fopen("Fraktal_CPU.txt","w");
    for(int y=0;y<DIM;y++){
      for(int x=0;x<DIM;x++){
        int offset=x+y*DIM;
        fprintf(fp, "\n%d, %d, %d, %d", ptr[offset*4+0], ptr[offset*4+1], ptr[offset*4+2], ptr[offset*4+3]);
      }
    }
  fclose(fp);
}

int main(void){
  DataBlock data;
  unsigned char *bitmap = (unsigned char*)malloc(DIM*DIM*4*sizeof(unsigned char));
  int image_size = DIM*DIM*4;
  kernel(bitmap);
  save_to_file (bitmap);
}
```

Wersja B – w CUDA C na GPU

```
%%cu
#include<stdio.h>
#define DIM 1000
struct cuComplex{
  float r;
  float i;
  //cuComplex(floata,floatb):r(a),i(b){}
  __device__ cuComplex(float a,float b):r(a),i(b){}
  __device__ float magnitude2(void){ return r*r+i*i;  }
  __device__ cuComplex operator*(const cuComplex &a){ return cuComplex(r*a.r-i*a.i,i*a.r+r*a.i);  }
  __device__ cuComplex operator+(const cuComplex &a){ return cuComplex(r+a.r,i+a.i);  }
};
__device__ int julia(int x,int y){
    const float scale=1.5;
    float jx=scale*(float)(DIM/2-x)/(DIM/2);
    float jy=scale*(float)(DIM/2-y)/(DIM/2);
    cuComplex c(-0.8,0.156);
    cuComplex a(jx,jy);
    int i=0;
    for(i=0;i<200;i++){
      a=a*a+c;
      if(a.magnitude2()>1000) return 0;
    }
    return 1;
}
```

```
__global__ void kernel(unsigned char *ptr){
    // Odwzorowanie z blockIdx na współrzędne piksela
    int x=blockIdx.x;
    int y=blockIdx.y;
    int offset=x+y*gridDim.x;

    // Obliczenie wartości dla tego punktu
    int juliaValue=julia(x,y);
    ptr[offset*4+0]=255*juliaValue;
    ptr[offset*4+1]=0;
    ptr[offset*4+2]=0;
    ptr[offset*4+3]=255;
}

//Dalsze elementy programu pozwalają za zapisanie Fraktala do pliku tekstowego
struct DataBlock{
  unsigned char *dev_bitmap;
};

//wypisywanie zawartosci bitmapy
void wypisz_RGB_fraktal(unsigned char *ptr){
  for(int y=0;y<DIM;y++){
    for(int x=0;x<DIM;x++){
      int offset=x+y*DIM;
      printf( "\n%d, %d, %d, %d", ptr[offset*4+0], ptr[offset*4+1], ptr[offset*4+2], ptr[offset*4+3]);
    }
    }
}
```

```
void save_to_file(unsigned char *ptr){
  FILE *fp=fopen("Fraktal_GPU.txt","w");
  for(int y=0;y<DIM;y++){
    for(int x=0;x<DIM;x++){
      int offset=x+y*DIM;
      fprintf(fp, "\n%d, %d, %d, %d", ptr[offset*4+0], ptr[offset*4+1], ptr[offset*4+2], ptr[offset*4+3]);
    }
  }
  fclose(fp);
}

int main(void){
    DataBlock data;
    unsigned char *bitmap = (unsigned char*)malloc(DIM*DIM*4*sizeof(unsigned char));
    int image_size = DIM*DIM*4;
    unsigned char *dev_bitmap;

    cudaMalloc((void**)&dev_bitmap, image_size);
    data.dev_bitmap=dev_bitmap;
    dim3 grid(DIM,DIM);
    kernel<<<grid,1>>>(dev_bitmap);
    cudaMemcpy(bitmap,dev_bitmap,image_size,cudaMemcpyDeviceToHost);
    cudaFree(dev_bitmap);
    save_to_file (bitmap);
}
```