Выполнила: Белоусова Е., ИП-911

Задание

Задание: реализовать код лекции 9 на CUDA C/C++. Сравнить производительность программ скомпилированных numba и nvcc.

Цель: получить начальные навыки работы с Numba python.

Задание: настроить среду для разработки OpenGL приложений (см. Лекция 10) и протестировать программу из лекции 10.

Цель: получить начальные навыки работы с OpenGL.

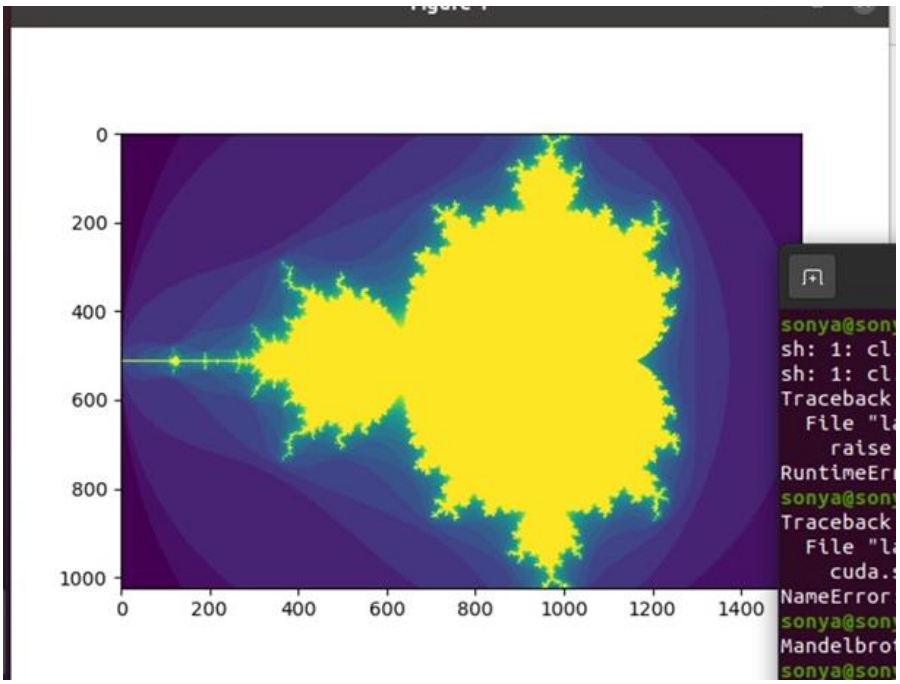
Описание работы программы

Сравним производительность программ, скомпилированных numba и nvcc.

```
(base) sonya@sonya-S551LB:~/cuda/lab9$ python3 lab85.py
/home/sonya/anaconda3/lib/python3.9/site-packages/numba/cuda/decorators.py:110:
NumbaDeprecationWarning: Eager compilation of device functions is deprecated (th
is occurs when a signature is provided)
  warn(NumbaDeprecationWarning(msg))
Mandelbrot created in 0.010480 s
(base) sonya@sonya-S551LB:~/cuda/lab9$ ./lab85
Mandelbrot created in 0.00123574 s
(base) sonya@sonya-S551LB:~/cuda/lab9$ python3 lab85_pycuda.py
```

	время	**ускорение**
numba jit	**0.010480**	**1**
nvcc	**0.00123**	**8,5**

Из таблицы видно, что код, скомпилированный nvcc работает быстрее почти в 8.5 раз.

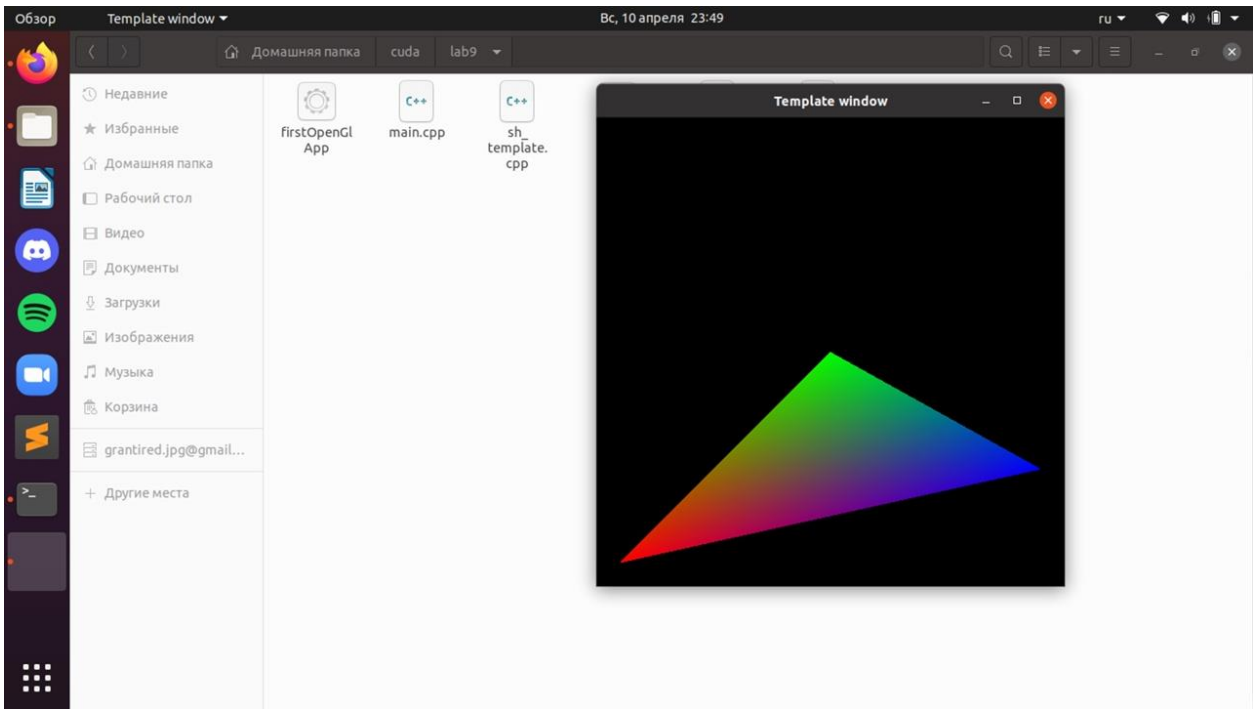


Настроим среду для разработки OpenGL приложений для Ubuntu 20.04 и протестируем программу из лекции 10.

Скомпилировать можно с помощью флагов –lGL –lGLU –lGLEW –lglfw.

Чтобы скопировать содержимое закадрового буфера в экранный буфер, вызовите SwapBuffers. Функция glSwapBuffers принимает дескриптор контекста устройства. Текущий формат пикселей для указанного контекста устройства должен включать обратный буфер. По умолчанию задний буфер находится за пределами экрана, а передний буфер - на экране.

В функции initBuffer(): с помощью glGenBuffers получаем идентификатор буффера, не указатель, просто число. Далее привязываем к ид тип буффера, его назначение. Объявляем на хосте массив и инициализируем вершины, положение и цвет. Далее выделяем память и копируем буффер с хоста на устройство. В display() нужно создать программы, передать им данные из буффера для отображения на экране.



Листинг

```
#include "cuda_runtime.h"

#include "device_launch_parameters.h"

#include <cufft.h>

#include <stdio.h>

#include <malloc.h>
```

```
#include <string.h>
#include <fstream>
#include <iostream>
#include <string>
#include <vector>
#include <algorithm>
#include <cctype>
#include <list>
#include <stdlib.h>
#include <ctime>

#include <cuComplex.h>

using namespace std;

#define IMG_WIDTH 1536
#define IMG_HEIGHT 1024

__device__ int mandel(float x, float y, unsigned int max_iters){
   cuFloatComplex c = make_cuFloatComplex(x, y);
   cuFloatComplex z = make_cuFloatComplex(0.0f, 0.0f);

   for(int i=0;i<max_iters;i++){
      z = cuCaddf(cuCmulf(z,z),c);
      if (z.x*z.x + z.y*z.y >= 4){
         return i;
      }
   }
   return max_iters;
}

__global__ void create_fractal(float min_x, float max_x, float min_y, float max_y, unsigned char* image,
unsigned int iters){
```

```
    int height = IMG_HEIGHT;
    int width = IMG_WIDTH;

    float pixel_size_x = (max_x - min_x)/ width;
    float pixel_size_y = (max_y - min_y)/ height;

    int startX = threadIdx.x+blockDim.x*blockIdx.x;
    int startY = threadIdx.y+blockDim.y*blockIdx.y;

    int gridX = gridDim.x * blockDim.x;
    int gridY = gridDim.y * blockDim.y;

    for(int x = startX; x<width; x+=gridX){
        float real = min_x + x * pixel_size_x;
        for(int y = startY;y<height; y+=gridY){
            float imag = min_y + y * pixel_size_y;
            image[x+y*width] = mandel(real,imag,iters);
        }
    }

}

int main(){
    cudaEvent_t start, stop;
    cudaEventCreate(&start);
    cudaEventCreate(&stop);
    float dt;

    unsigned char* image = (unsigned char*)calloc(IMG_WIDTH*IMG_HEIGHT,sizeof(unsigned char));
    unsigned char* d_image;

    cudaMalloc((void**)&d_image, sizeof(unsigned char) * IMG_WIDTH * IMG_HEIGHT);
```

```
   dim3 blockdim(32,8);
   dim3 griddim(32,16);

   cudaMemcpy(d_image,image,sizeof(unsigned char) * IMG_WIDTH *
IMG_HEIGHT,cudaMemcpyHostToDevice);

   cudaEventRecord(start, 0);

   create_fractal<<<griddim, blockdim>>>(-2.0, 1.0, -1.0, 1.0, d_image, 20);

   cudaEventRecord(stop, 0);
   cudaEventSynchronize(stop);
   cudaEventElapsedTime(&dt, start, stop);

   cudaMemcpy(image,d_image,sizeof(unsigned char) * IMG_WIDTH *
IMG_HEIGHT,cudaMemcpyDeviceToHost);

   cout << "Mandelbrot created in " << (dt/1000) << " s \n";
   return 0;
}
```