Федеральное государственное бюджетное
образовательное учреждение высшего образования
«Сибирский государственный университет
телекоммуникаций и информатики» (СибГУТИ)

09.03.01 "Информатика и вычислительная техника"
профиль "Программное обеспечение средств
вычислительной техники и автоматизированных систем"

ОТЧЕТ
по дисциплине «Программирование графических процессоров»
лабораторная работа 8.5

Выполнил:
Разумов Д. Б.
студент группы ИП-811

Проверил:
преподаватель Малков Е.А.

Новосибирск 2021

Оглавление

Задание

Задание: написать две программы, которые изображают множество Мандельброта. Первая программа должна быть написана на python с использованием numba и CUDA. Вторая — на чистом CUDA C. Необходимо сравнить производительность этих программ.

Листинг

Файл python_numba_cuda.py

```
import numpy as np
from pylab import imshow, show
from timeit import default_timer as timer
from numba import cuda
from numba import *

@cuda.jit(uint32(f8, f8, uint32), device=True)
def mandel(x, y, max_iters):
    c = complex(x, y)
    z = 0.0j
    for i in range(max_iters):
        z = z*z + c   #отображение Мандельброта
        if (z.real*z.real + z.imag*z.imag) >= 4:
            return i
    return max_iters

@cuda.jit((f8, f8, f8, f8, uint8[:,:], uint32))
def create_fractal(min_x, max_x, min_y, max_y, image, iters):
    height = image.shape[0] #размерности двумерного массива
    width = image.shape[1]

    pixel_size_x = (max_x - min_x) / width #задание размеров пикселя
    pixel_size_y = (max_y - min_y) / height
```

```
    startX, startY = cuda.grid(2) #threadIdx.x+blockDim.x*blockIdx.x,...
    gridX = cuda.gridDim.x * cuda.blockDim.x;
    gridY = cuda.gridDim.y * cuda.blockDim.y;

    for x in range(startX, width, gridX): #если width>gridX
        real = min_x + x * pixel_size_x
        for y in range(startY, height, gridY):
            imag = min_y + y * pixel_size_y
            image[y, x] = mandel(real, imag, iters)

image = np.zeros((1024, 1536), dtype = np.uint8) #инициализация массива

blockdim = (32, 8)
griddim = (32,16)

d_image = cuda.to_device(image)
create_fractal[griddim, blockdim](-2.0, 1.0, -1.0, 1.0, d_image, 20)
#d_image.to_host()

start = timer()
create_fractal[blockdim, griddim](-2.0, 1.0, -1.0, 1.0, image, 20)
dt = timer() - start
print ("Mandelbrot created in %f s" % dt)
imshow(image)
show()
```

Файл cuda_c.cu

```
#include <stdio.h>
//#include <cuda_runtime.h>
#include <inttypes.h> //для использования uint8_t

#define CUDA_CHECK_RETURN(value) {\
	cudaError_t _m_cudaStat = value;\
	if (_m_cudaStat != cudaSuccess) {\
		fprintf(stderr, "Error \"%s\" at line %d in file %s\n",\
			cudaGetErrorString(_m_cudaStat), __LINE__, __FILE__);\
		exit(1);\
	}\
}

struct complex { //структура для комлексного числа
	float real; //действительная часть
	float imag; //мнимая часть
};

__device__ struct complex complex_mul(struct complex z1, struct complex z2)
{
	struct complex rez;
	rez.real = z1.real * z2.real - z1.imag * z2.imag;
	rez.imag = z1.real * z2.imag + z2.real * z1.imag;
	return rez;
}
__device__ struct complex complex_sum(struct complex z1, struct complex z2)
```

```
{
	struct complex rez;
	rez.real = z1.real + z2.real;
	rez.imag = z1.imag + z2.imag;
	return rez;
}

__device__ uint8_t mandel(float x, float y, int max_iters) {
	struct complex c; //c = complex(x, y)
	c.real = x, c.imag = y;
	struct complex z; //z = 0.0j
	z.real = z.imag = 0.0f;
	for (int i = 0; i < max_iters; i++) {
		z = complex_mul(z, z); //z = z*z + c; //отображение Мандельброта
		z = complex_sum(z, c);
		if ((z.real * z.real + z.imag * z.imag) >= 4)
			return i;
	}
	return max_iters;
}

__global__ void create_fractal_dev(float min_x, float max_x, float min_y, float
max_y,
	uint8_t *image, int height, int width, int iters)
{
	float pixel_size_x = (max_x - min_x) / (width); //задание размеров пикселя
	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;
			uint8_t color = mandel(real, imag, iters);
			image[x + y * width] = color; //задание цвета пикселя
		}
	}
}

int main() {
	/// размерности массивов:
	int N = 1024, M = 1536; //размерности массива
	const size_t size_in_bytes = N * M * sizeof(uint8_t);

	/// создание массивов:
	uint8_t *A_dev;
	CUDA_CHECK_RETURN(cudaMalloc( (void **) &A_dev, size_in_bytes));
	cudaMemset(A_dev, 0, size_in_bytes); //заполнить нулями
	uint8_t *A_hos;
	A_hos = (uint8_t*) malloc(size_in_bytes);
	cudaMemset(A_hos, 0, size_in_bytes); //заполнить нулями
```

```
	/// как расспараллелить код
	dim3 dimBlock(32, 8); //число выделенных блоков
	dim3 dimGrid(32,16); //размер и размерность сетки

	/// создание CUDA-событий
	cudaEvent_t start, stop;
	CUDA_CHECK_RETURN(cudaEventCreate(&start));
	CUDA_CHECK_RETURN(cudaEventCreate(&stop));

	/// запуск ядра
	cudaEventRecord(start, 0);
		create_fractal_dev <<<dimGrid,dimBlock >>> (-2.0, 1.0, -1.0, 1.0, A_dev, N, M, 20);
	cudaEventRecord(stop, 0);
	cudaEventSynchronize(stop);

	/// время
	float time;
	cudaEventElapsedTime(&time, start, stop);
	printf("time = %f s\n", time / 1000);

	/// запись массива в файл
	cudaMemcpy(A_hos, A_dev, size_in_bytes, cudaMemcpyDeviceToHost);
	FILE *fp = fopen("rez.dat", "w");
	for (int i = 0; i < N; i++) {
		for (int j = 0; j < M; j++) {
			fprintf(fp, "%d ", A_hos[i * M + j]);
```

```
            }
            fprintf(fp, "\n");
        }
        fclose(fp);

        /*
        struct complex z1, z2;
        z1.real = 3; z1.imag = 1;
        z2.real = 2; z2.imag = -3;
        struct complex rez = complex_mul(z1, z2);
        printf("%f %f \n", rez.real, rez.imag);
        rez = complex_sum(z1, z2);
        printf("%f %f \n", rez.real, rez.imag);
        */
        return 0;
}
```

Скриншоты

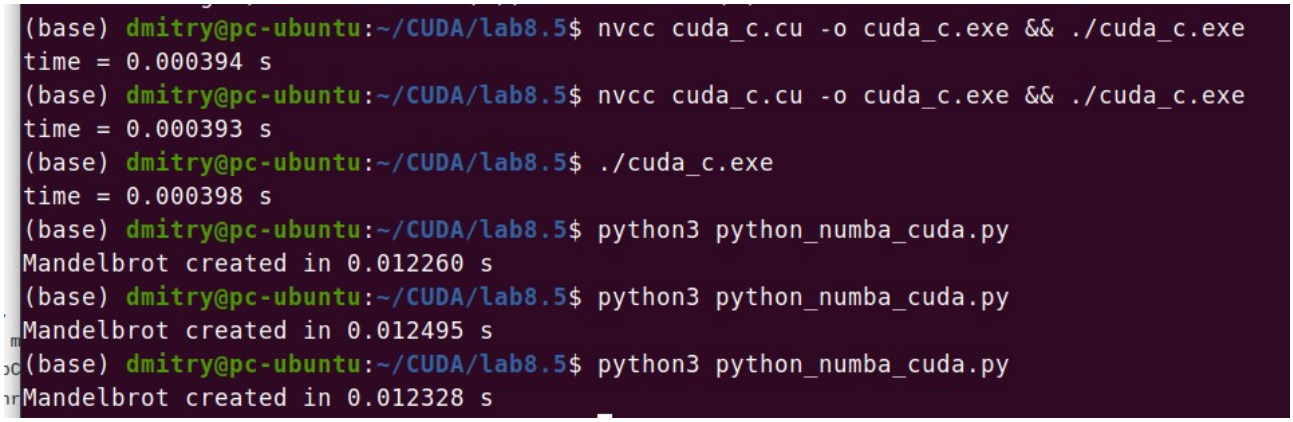

Рисунок 1 — сравнение производительности

Об используемом GPU

Device name: GeForce MX350

Global memory available on device: 2099904512 (2002 MByte)

Shared memory available per block: 49152

Count of 32-bit registers available per block: 65536

Warp size in threads: 32

Maximum pitch in bytes allowed by memory copies: 2147483647

Maximum number of threads per block: 1024

Maximum size of each dimension of a block[0]: 1024

Maximum size of each dimension of a block[1]: 1024

Maximum size of each dimension of a block[2]: 64

Maximum size of each dimension of a grid[0]: 2147483647

Maximum size of each dimension of a grid[1]: 65535

Maximum size of each dimension of a grid[2]: 65535

Clock frequency in kilohertz: 1468000

totalConstMem: 65536

Major compute capability: 6

Minor compute capability: 1

Number of multiprocessors on device: 5

Count of cores: 640

Id current device: 0

Id nearest device to 1.3: 0

Заключение

Программа, написанная на CUDA C оказалась быстрее программы, написанной на Python numba CUDA.