

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 7

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

Проверил:  
преподаватель Малков Е.А.

# Оглавление

Задание.....3

Листинг.....4

Скриншоты.....10

Об используемом GPU.....11

Заключение.....12

# Задание

## Задание:

- программно реализовать алгоритм решения уравнения переноса  $\frac{\partial f}{\partial t} + u \frac{\partial f}{\partial x} = 0$  на основе разностной схемы «вверх по потоку» -  $f_i^{n+1} = f_i^n + \frac{u \tau}{h} (f_{i-1}^n - f_i^n), u > 0$  с использованием библиотеки *Thrust* и без её использования (сырой код *CUDA C*).
- Сравнить производительность реализаций.

**Цель:** получить навыки использования библиотеки *Thrust*.

## ЛИСТИНГ

```
#include <iostream>
#include <cmath>
#include <stdio.h>
#include <thrust/device_vector.h>
#include <thrust/fill.h>
#include <thrust/host_vector.h>
#include <thrust/sequence.h>
#include <thrust/transform.h>
using namespace std;

#define A 0.2
#define B 0.01

__global__ void kernel(float coeff, float* f, float* res)
{
    int curr = threadIdx.x + blockDim.x * blockIdx.x;
    int prev = curr - 1;
    if (prev == -1) {
        res[curr] = f[curr];
    }
    else {
        res[curr] = f[curr] + (coeff) * (f[prev] - f[curr]);
    }
}
```

```

struct Functor
{
    const float coeff;
    Functor(float _coeff) : coeff(_coeff) {}
    __host__ __device__ float operator()(float x, float y) {
        return x + coeff * (y - x);
    }
};

void iteration(float _coeff, thrust::device_vector<float>::iterator x,
              thrust::device_vector<float>::iterator xs,
              thrust::device_vector<float>::iterator y)
{
    Functor func(_coeff);
    thrust::transform(x + 1, xs, x, y + 1, func);
}

float x_func(float x)
{
    return x * x * exp(-(x - A) * (x - A) / B);
}

float t_func(float t)
{
    return 0;
}

```

```

int main()
{
    cudaDeviceProp deviceProp;
    cudaGetDeviceProperties(&deviceProp, 0);
    printf("device: %s \n\n", deviceProp.name);

    int Nx = 8192;
    int Nt = 8192;
    float tl = 0.2;

    float dx = 1.0f / Nx;
    float dt = tl / Nt;

    cudaEvent_t start, stop;
    float time;
    cudaEventCreate(&start);
    cudaEventCreate(&stop);

    float *x, *t;
    thrust::host_vector<float> thr(Nx * Nt);
    float* cda;

    cudaHostAlloc((void**)&x, Nx * sizeof(float), cudaHostAllocDefault);
    cudaHostAlloc((void**)&t, Nt * sizeof(float), cudaHostAllocDefault);
    cudaHostAlloc((void**)&cda, Nt * Nx * sizeof(float), cudaHostAllocDefault);
    for (int i = 0; i < Nx; i++) {

```

```

        for (int j = 0; j < Nt; j++) {
            thr[i + j * Nt] = 0;
            cda[i + j * Nt] = 0;
        }
    }
}

```

```

float value = 0;
for (int i = 0; i < Nx; i++, value += dx) {
    x[i] = value;
    thr[i] = x_func(x[i]);
    cda[i] = x_func(x[i]);
}

```

```

value = 0;
for (int i = 0; i < Nt; i++, value += dt) {
    t[i] = value;
    thr[i * Nt] = t_func(t[i]);
    cda[i * Nt] = t_func(t[i]);
}

```

```

thrust::device_vector<float> dev(Nx * Nt);
thrust::copy(thr.begin(), thr.end(), dev.begin());
Functor func(dt / dx);

```

```

/// thrust:

```

```

cudaEventRecord(start, 0);

```

```

for (int j = 0; j < Nt - 1; j++) {

```

```

    thrust::transform(dev.begin() + (j * Nx) + 1, dev.begin() + ((j + 1) * Nx),

```

```

dev.begin() + (j * Nx), dev.begin() + ((j + 1) * Nx)
+ 1,

func);

}

cudaEventRecord(stop, 0);

cudaEventSynchronize(stop);

thrust::copy(dev.begin(), dev.end(), thr.begin()); //скопировать с устройства
на хост

cudaEventElapsedTime(&time, start, stop);

cout << "Thrust time is " << time << "ms\n";

/// обычный CUDA:

float* dev_cda;

cudaMalloc((void **)&dev_cda, Nx * Nt * sizeof(float));

cudaMemcpy(dev_cda, cda, Nx * Nt * sizeof(float),
cudaMemcpyHostToDevice);

cudaEventRecord(start, 0);

for (int i = 0; i < Nt - 1; i++) {

    kernel <<< Nx / 256, 256 >>> (dt / dx, dev_cda + (i * Nx), dev_cda + ((i
+ 1) * Nx));

    cudaDeviceSynchronize();

}

cudaEventRecord(stop, 0);

cudaEventSynchronize(stop);

    cudaMemcpy(cda, dev_cda, Nx * Nt * sizeof(float),
cudaMemcpyDeviceToHost);

cudaEventElapsedTime(&time, start, stop);

cout << "CUDA time is " << time << "ms\n\n";

```



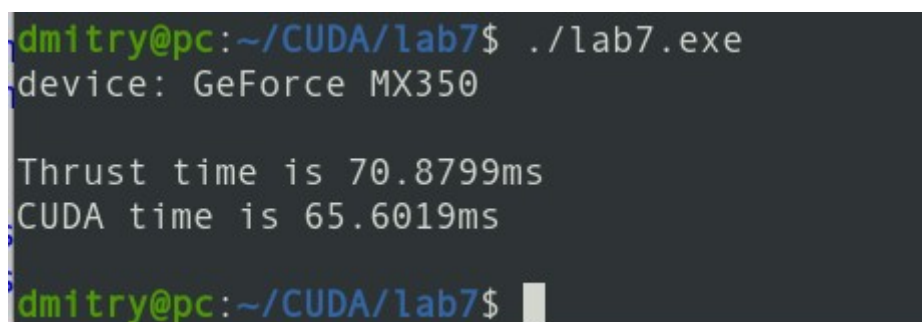
```

/*
/// проверка:
float thr_sum = 0.0f;
float cda_sum = 0.0f;
for (int i = 0; i < Nx; i++) {
    for (int j = 0; j < Nt; j++) {
        cda_sum += cda[i + j * Nt];
        thr_sum += thr[i + j * Nt];
    }
}
printf("thr_sum = %f \ncda_sum = %f \n", thr_sum, cda_sum);
*/

cudaFree(dev_cda);
cudaEventDestroy(start);
cudaEventDestroy(stop);
return 0;
}

```

## Скриншоты

A screenshot of a terminal window with a dark background. The text is as follows:

```
dmitry@pc:~/CUDA/lab7$ ./lab7.exe
device: GeForce MX350

Thrust time is 70.8799ms
CUDA time is 65.6019ms

dmitry@pc:~/CUDA/lab7$
```

Рисунок 1 — запуск программы

## 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

Код, выполненный без использования Thrust, оказался быстрее примерно на 12%.

Библиотека Thrust упрощает написание кода, однако некоторые задачи невозможно выполнить используя исключительно Thrust.