

Выполнила: Белоусова Е., ИП-911

Задание:

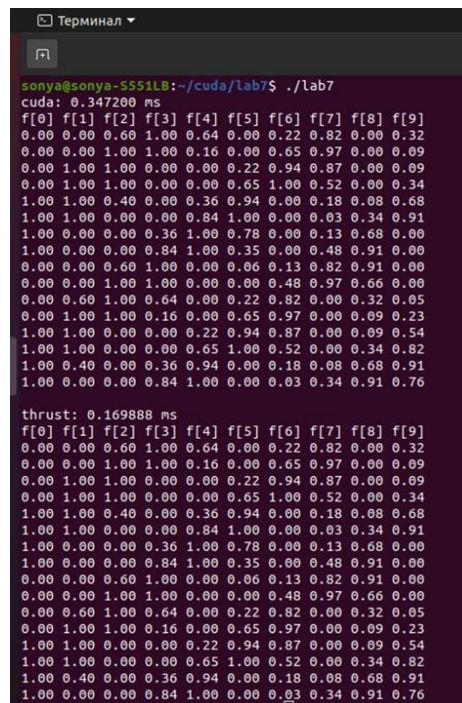
- программно реализовать алгоритм решения уравнения переноса  $\frac{\partial f}{\partial t} + u \frac{\partial f}{\partial x} = 0$  на основе разностной схемы «вверх по потоку» -  $f_i^{n+1} = f_i^n + \frac{u \tau}{h} (f_{i-1}^n - f_i^n), u > 0$  с использованием библиотеки *Thrust* и без её использования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

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

**Примечание:** детали задания разъясняются на практическом занятии.

## Описание работы программы

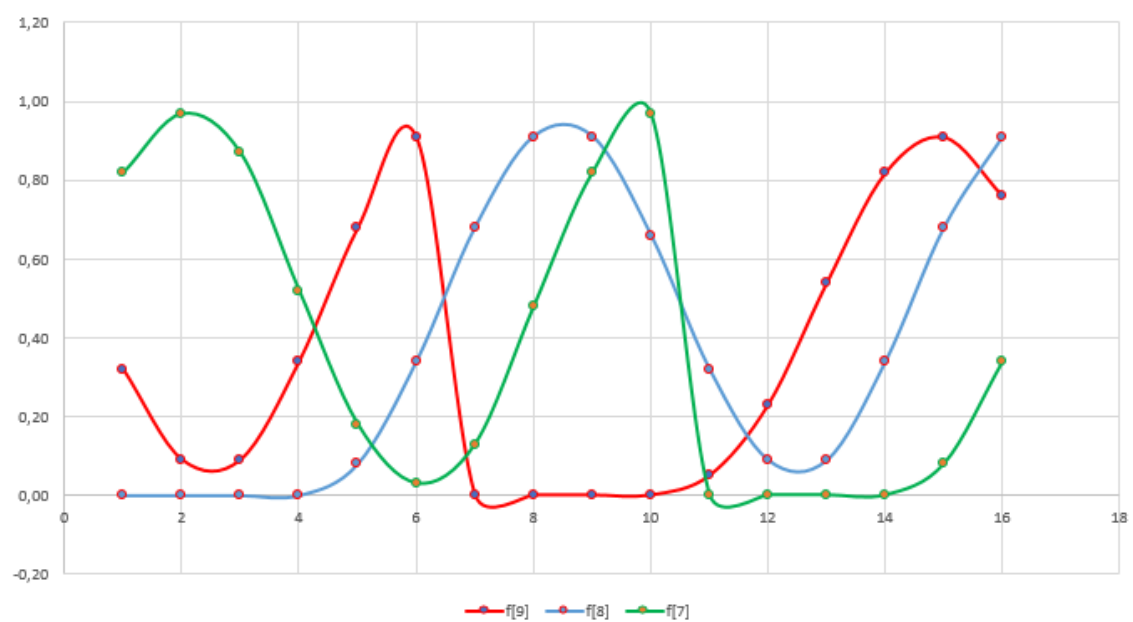
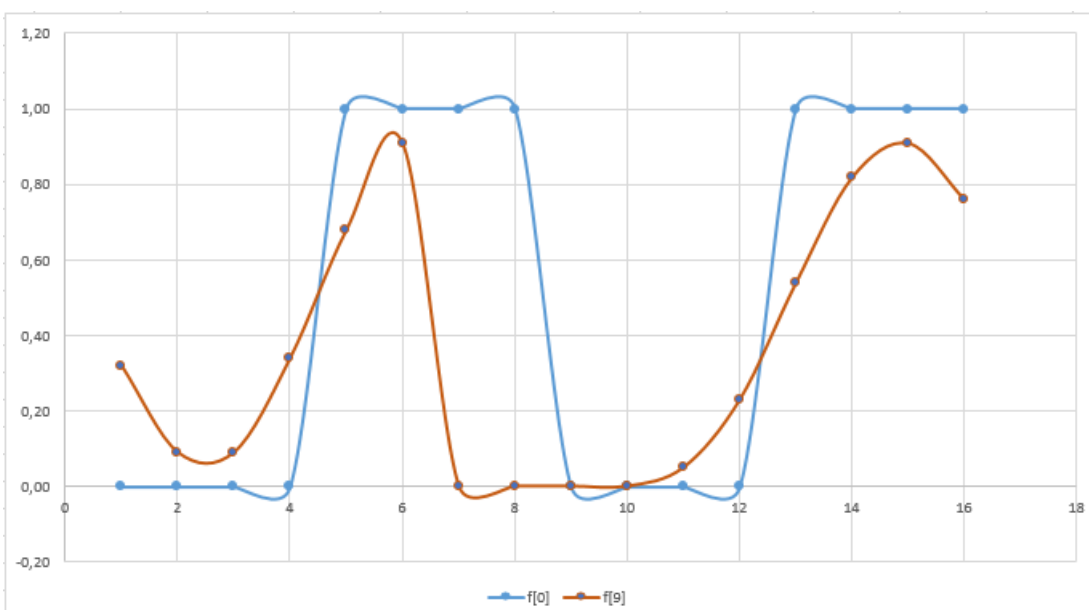
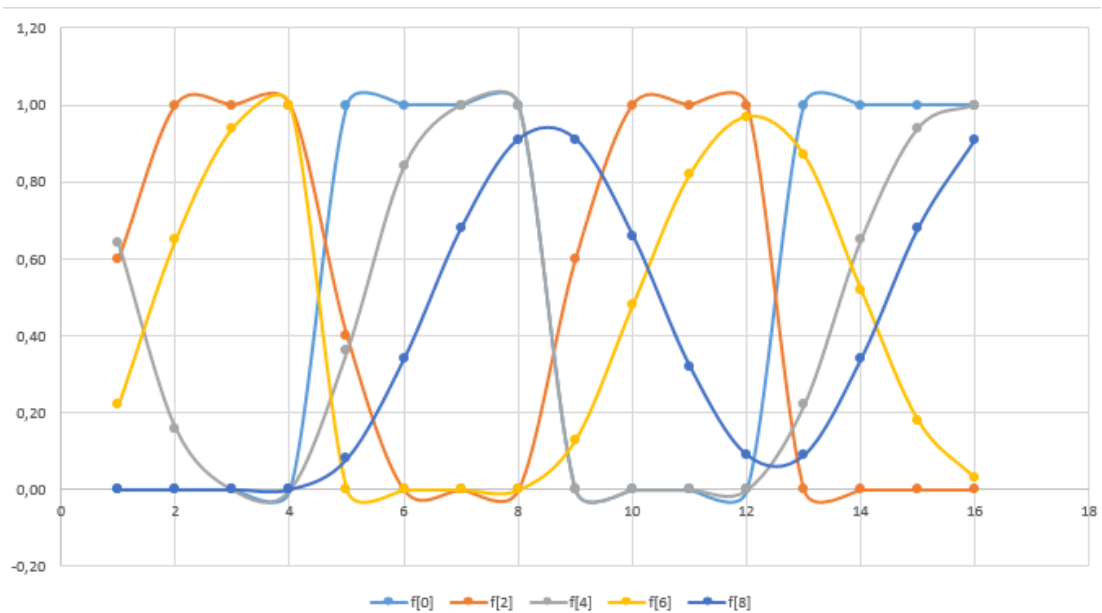
Для выполнения задания с помощью библиотеки *ehrust*, проинициализируем вектор на хосте, затем объявим вектор на девайсе и скопируем в него данные с помощью `thrust::copy`. Далее используем алгоритм `thrust::transform`, который выполнит операцию, определенную в функторе над входными значениями и запишет результат выполнения по заданному итератору. Реализуем алгоритм решения уравнения переноса на *CUDA C*.



```
Терминал
sonya@sonya-S551LB:~/cuda/lab7$ ./lab7
cuda: 0.347200 ms
f[0] f[1] f[2] f[3] f[4] f[5] f[6] f[7] f[8] f[9]
0.00 0.00 0.60 1.00 0.64 0.00 0.22 0.82 0.00 0.32
0.00 0.00 1.00 1.00 0.16 0.00 0.65 0.97 0.00 0.09
0.00 1.00 1.00 0.00 0.00 0.22 0.94 0.87 0.00 0.09
0.00 1.00 1.00 0.00 0.00 0.65 1.00 0.52 0.00 0.34
1.00 1.00 0.40 0.00 0.36 0.94 0.00 0.18 0.08 0.68
1.00 1.00 0.00 0.00 0.84 1.00 0.00 0.03 0.34 0.91
1.00 0.00 0.00 0.36 1.00 0.78 0.00 0.13 0.68 0.00
1.00 0.00 0.00 0.84 1.00 0.35 0.00 0.48 0.91 0.00
0.00 0.00 0.60 1.00 0.00 0.06 0.13 0.82 0.91 0.00
0.00 0.00 1.00 1.00 0.00 0.00 0.48 0.97 0.66 0.00
0.00 0.60 1.00 0.64 0.00 0.22 0.82 0.00 0.32 0.05
0.00 1.00 1.00 0.16 0.00 0.65 0.97 0.00 0.09 0.23
1.00 1.00 0.00 0.00 0.22 0.94 0.87 0.00 0.09 0.54
1.00 1.00 0.00 0.00 0.65 1.00 0.52 0.00 0.34 0.82
1.00 0.40 0.00 0.36 0.94 0.00 0.18 0.08 0.68 0.91
1.00 0.00 0.00 0.84 1.00 0.00 0.03 0.34 0.91 0.76

thrust: 0.169888 ms
f[0] f[1] f[2] f[3] f[4] f[5] f[6] f[7] f[8] f[9]
0.00 0.00 0.60 1.00 0.64 0.00 0.22 0.82 0.00 0.32
0.00 0.00 1.00 1.00 0.16 0.00 0.65 0.97 0.00 0.09
0.00 1.00 1.00 0.00 0.00 0.22 0.94 0.87 0.00 0.09
0.00 1.00 1.00 0.00 0.00 0.65 1.00 0.52 0.00 0.34
1.00 1.00 0.40 0.00 0.36 0.94 0.00 0.18 0.08 0.68
1.00 1.00 0.00 0.00 0.84 1.00 0.00 0.03 0.34 0.91
1.00 0.00 0.00 0.36 1.00 0.78 0.00 0.13 0.68 0.00
1.00 0.00 0.00 0.84 1.00 0.35 0.00 0.48 0.91 0.00
0.00 0.00 0.60 1.00 0.00 0.06 0.13 0.82 0.91 0.00
0.00 0.00 1.00 1.00 0.00 0.00 0.48 0.97 0.66 0.00
0.00 0.60 1.00 0.64 0.00 0.22 0.82 0.00 0.32 0.05
0.00 1.00 1.00 0.16 0.00 0.65 0.97 0.00 0.09 0.23
1.00 1.00 0.00 0.00 0.22 0.94 0.87 0.00 0.09 0.54
1.00 1.00 0.00 0.00 0.65 1.00 0.52 0.00 0.34 0.82
1.00 0.40 0.00 0.36 0.94 0.00 0.18 0.08 0.68 0.91
1.00 0.00 0.00 0.84 1.00 0.00 0.03 0.34 0.91 0.76
```

По результатам выполнения, можно заметить, что с помощью библиотеки *thrust*, алгоритм выполняется быстрее почти в 2 раза.



## Листинг

```
#include "cuda_runtime.h"

#include "device_launch_parameters.h"

#include <iostream>

#include <thrust/device_vector.h>

#include <thrust/fill.h>

#include <thrust/host_vector.h>

#include <thrust/sequence.h>

#include <thrust/transform.h>

#include <stdio.h>


using namespace std;


#define step 4

#define N 16

#define Nt 10

#define ALPHA 0.2

#define T 2

struct functor
{
    const float koef;

    functor(float _koef) : koef(_koef){}

    __host__ __device__ float operator()(float x, float y)
    {
        return x + koef * (y - x);
    }
};


__global__ void kernel(float *f, float *res)
{
    int cur = threadIdx.x + blockDim.x * blockIdx.x;
```

```

int prev = cur - 1;
if(prev == -1)
{
    res[cur] = f[cur];
}else
{
    res[cur] = f[cur] + ALPHA * T * (f[prev] - f[cur]);
}
}

int data(int curr)
{
    if(curr < 4 || curr >= 8 && curr < 12) return 0;
    else return 1;
}

int main()
{
    float funct[N * Nt];
    float funcData[N * Nt];
    float *temp;

    cudaEvent_t start, stop;
    float time;

    cudaEventCreate(&start);
    cudaEventCreate(&stop);
    for(int i = 0; i < N; i++) {
        for(int j = 0; j < Nt; j++) {
            funcData[i + j * Nt] = 0;
        }
    }
}

```

```

for(int i = 0; i < N; i++) {
    funcData[i + 0 * Nt] = data(i);
}

cudaMalloc((void **)&temp, sizeof(float) * N * Nt);
cudaMemcpy(temp, funcData, sizeof(float) * N * Nt, cudaMemcpyHostToDevice);

cudaEventSynchronize(start);
cudaEventRecord(start, 0);
for (int i = 0; i < 10; i++) {
    kernel <<< 1, N >>> (temp + (i * N), temp + ((i + 1) * N));
    cudaDeviceSynchronize();
}
cudaDeviceSynchronize();
cudaEventRecord(stop, 0);
cudaEventSynchronize(stop);
cudaEventElapsedTime(&time, start, stop);
cudaMemcpy(funcnt, temp, N * Nt * sizeof(float), cudaMemcpyDeviceToHost);
printf("cuda: %f ms\n", time);
for(int i = 0; i < Nt; i++) {
    printf("f[%d] ", i);
}
printf("\n");
for(int i = 0; i < N; i++) {
    for(int j = 0; j < Nt; j++) {
        printf("%.2f ", funcnt[i + j * Nt]);
    }
    printf("\n");
}

thrust::host_vector<float> vect(N * 10);

```

```

    for (int i = 0; i < N; i++)
    {
        vect[i] = funcData[i];
    }

    thrust::device_vector<float> x(N * 10);
    thrust::copy(vect.begin(), vect.end(), x.begin());

    cudaEventSynchronize(start);
    cudaEventRecord(start, 0);

    functor func(ALPHA * T);

    for(int j = 0; j < 10; j++){
        thrust::transform(x.begin()+(j*N)+1, x.begin()+((j+1)*N), x.begin()+(j*N), x.begin()+((j+1)*N)+1,
func);
    }

    thrust::copy(x.begin(), x.end(), vect.begin());

    cudaDeviceSynchronize();
    cudaEventRecord(stop, 0);
    cudaEventSynchronize(stop);
    cudaEventElapsedTime(&time, start, stop);
    std::cout << std::endl;
    printf("thrust: %f ms\n", time);

    for(int i = 0; i < Nt; i++)
    {
        printf("f[%d] ", i);

    }

    printf("\n");

    for(int i = 0; i < N; i++){
for(int j = 0; j < Nt; j++){
        printf("%.2f ", vect[i + j * Nt]);
    }

    printf("\n");
}

return 0;

```

