//Подключение библиотек Thrust и CuBLAS:

#include <thrust/host\_vector.h>

#include <thrust/device\_vector.h>

#include <thrust/transform.h>

#include <thrust/fill.h>

#include <thrust/sequence.h>

#include <stdio.h>

#include <cublas\_v2.h>

//Макрос для определения ошибки:

//cudaGetErrorString – возвращает сообщение с кодом ошибки

//\_\_LINE\_\_, \_\_FILE\_\_ - предопределённыые макросы препроцессора для определения местоположения в коде программы – номер строки и имени файла.

#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);\

}\

}

//Реализация saxpy

//Функтор Saxpy:

struct saxpy\_functor {

const float a;

saxpy\_functor(float \_a) : a(\_a) {}

\_\_host\_\_ \_\_device\_\_ float operator()(float x, float y) {

return a \* x + y;

}

};

//Алгоритм transform:

void saxpy(float a, thrust::device\_vector<float>& x, thrust::device\_vector<float>& y) {

saxpy\_functor func(a);

thrust::transform(x.begin(), x.end(), y.begin(), y.begin(), func);

}

\_\_global\_\_ void gSaxpy(float alpha, float \*x, float \*y) {

int i = threadIdx.x + blockIdx.x \* blockDim.x;

y[i] = alpha \* x[i] + y[i];

}

//Реализация «сырого» CUDA C кода:

float saxpyCUDAC() {

cudaEvent\_t start, stop;

float \*x\_d, \*x\_h, \*y\_h, \*y\_d;

float elapsedTime;

//Создание объекта событий start и stop:

CUDA\_CHECK\_RETURN(cudaEventCreate(&start));

CUDA\_CHECK\_RETURN(cudaEventCreate(&stop));

//Выделяем память на устройствах:

//cudaMalloc(указатель на выделенную память устройства, запрошенный размер выделения в байтах)

CUDA\_CHECK\_RETURN(cudaMalloc((void\*\*)&x\_d, N \* sizeof(float)));

CUDA\_CHECK\_RETURN(cudaMalloc((void\*\*)&y\_d, N \* sizeof(float)));

x\_h = (float\*)calloc(N, sizeof(float));

y\_h = (float\*)calloc(N, sizeof(float));

for(int i = 0; i < N; i++) {

x\_h[i] = i;

y\_h[i] = 0.87;

}

//Копируем данные между хостом и устройством:

//cudaMemcpy(адрес памяти назначения, адрес источника памяти, размер в байтах для копирования, тип перевода)

CUDA\_CHECK\_RETURN(cudaMemcpy(x\_d, x\_h, N \* sizeof(float), cudaMemcpyHostToDevice));

CUDA\_CHECK\_RETURN(cudaMemcpy(y\_d, y\_h, N \* sizeof(float), cudaMemcpyHostToDevice));

//Записывает событие start:

cudaEventRecord(start, 0);

gSaxpy <<< N / 256, 256 >>> (3.0, x\_d, y\_d);

//Ожидание окончания работы вычислительного устройства, возвращает ошибку, если одна из предыдущих задач не выполнена:

CUDA\_CHECK\_RETURN(cudaDeviceSynchronize());

//Записывает событие stop:

cudaEventRecord(stop, 0);

//Ожидает завершения события stop:

cudaEventSynchronize(stop);

//Вычисляет прошедшее время между событиями:

cudaEventElapsedTime(&elapsedTime, start, stop);

//Копируем данные с устройства на хост:

CUDA\_CHECK\_RETURN(cudaMemcpy(y\_h, y\_d, N \* sizeof(float), cudaMemcpyDeviceToHost));

cudaEventDestroy(start);

cudaEventDestroy(stop);

//Освобождает память на устройстве:

cudaFree(x\_d);

cudaFree(y\_d);

free(x\_h);

free(y\_h);

return elapsedTime;

}

//Реализация алгоритма линейной алгебры с использование библиотеки CuBLAS:

float saxpyCuBLAS() {

cudaEvent\_t start, stop;

float \*x\_h, \*y\_h, \*x\_d, \*y\_d;

float elapsedTime;

//Создание объекта событий start и stop:

CUDA\_CHECK\_RETURN(cudaEventCreate(&start));

CUDA\_CHECK\_RETURN(cudaEventCreate(&stop));

//Выделяем заблокированную страницу памяти на хосте:

//cudaMallocHost(указатель на выделенную память хоста, запрошенный размер выделения в байтах)

CUDA\_CHECK\_RETURN(cudaMallocHost((void\*\*)&x\_h, N \* sizeof(float)));

CUDA\_CHECK\_RETURN(cudaMallocHost((void\*\*)&y\_h, N \* sizeof(float)));

//Выделяем память на устройствах:

//cudaMalloc(указатель на выделенную память устройства, запрошенный размер выделения в байтах)

CUDA\_CHECK\_RETURN(cudaMalloc((void\*\*)&x\_d, N \* sizeof(float)));

CUDA\_CHECK\_RETURN(cudaMalloc((void\*\*)&y\_d, N \* sizeof(float)));

for(int i = 0; i < N; i++) {

x\_h[i] = (float) i;

y\_h[i] = 0.87f;

}

//Инициализация библиотеки CUBLAS:

cublasHandle\_t cublas\_handle;

cublasCreate(&cublas\_handle);

const int num\_rows = N;

const int num\_cols = 1;

const size\_t elem\_size = sizeof(float);

//Копирование матрицы с числом строк num\_elem и одним столбцом с хоста на устройство:

cublasSetMatrix(num\_rows, num\_cols, elem\_size, x\_h, num\_rows, x\_d, num\_rows);

cublasSetMatrix(num\_rows, num\_cols, elem\_size, y\_h, num\_rows, y\_d, num\_rows);

const int stride = 1;

float alpha = 3.0f;

//Записывает событие start:

CUDA\_CHECK\_RETURN(cudaEventRecord(start, 0));

//Выполнение SingleAlphaXPlusY:

cublasSaxpy(cublas\_handle, N, &alpha, x\_d, stride, y\_d, stride);

//Записывает событие stop:

CUDA\_CHECK\_RETURN(cudaEventRecord(stop, 0));

//Ожидает завершения события stop:

CUDA\_CHECK\_RETURN(cudaEventSynchronize(stop));

//Вычисляет прошедшее время между событиями:

CUDA\_CHECK\_RETURN(cudaEventElapsedTime(&elapsedTime, start, stop));

//Копирование матриц с числом строк num\_elem и одним столбцом с устройства на хост:

cublasGetMatrix(num\_rows, num\_cols, elem\_size, x\_d, num\_rows, x\_h, num\_rows);

cublasGetMatrix(num\_rows, num\_cols, elem\_size, y\_d, num\_rows, y\_h, num\_rows);

cudaEventDestroy(start);

cudaEventDestroy(stop);

//Освобождаем ресурсы на устройстве:

cublasDestroy(cublas\_handle);

CUDA\_CHECK\_RETURN(cudaFree(x\_d));

CUDA\_CHECK\_RETURN(cudaFree(y\_d));

//Освобождаем ресурсы на хосте:

CUDA\_CHECK\_RETURN(cudaFreeHost(x\_h));

CUDA\_CHECK\_RETURN(cudaFreeHost(y\_h));

return elapsedTime;

}

//Реализация алгоритма линейной алгебры с использование библиотеки Thrust:

float saxpyTHRUST() {

cudaEvent\_t start, stop;

float elapsedTime;

//Создание объекта событий start и stop:

CUDA\_CHECK\_RETURN(cudaEventCreate(&start));

CUDA\_CHECK\_RETURN(cudaEventCreate(&stop));

//Контейнеры host-vector:

//Инициализируем host\_vector N числом:

thrust::host\_vector<float> h1(N);

thrust::host\_vector<float> h2(N);

//Устанавливаем все элементы h1 в 0, 1, 2, 3:

thrust::sequence(h1.begin(), h1.end());

//Устанавливаем элементы вектора в 0.87:

thrust::fill(h2.begin(), h2.end(), 0.87);

//Контейнеры device\_vector:

thrust::device\_vector<float> d1 = h1;

thrust::device\_vector<float> d2 = h2;

//Записывает события start и stop:

CUDA\_CHECK\_RETURN(cudaEventRecord(start, 0));

saxpy(3.0, d1, d2);

CUDA\_CHECK\_RETURN(cudaEventRecord(stop, 0));

//Ожидает завершения события stop:

CUDA\_CHECK\_RETURN(cudaEventSynchronize(stop));

//Вычисляет прошедшее время между событиями:

CUDA\_CHECK\_RETURN(cudaEventElapsedTime(&elapsedTime, start, stop));

h2 = d2;

h1 = d1;

cudaEventDestroy(start);

cudaEventDestroy(stop);

return elapsedTime;

}

int main() {

float elapsedTime;

//Вычисление времени выполнения «сырого» CUDA C кода при реализации алгоритма линейной алгебры:

elapsedTime = saxpyCUDAC();

printf("CUDA C Time: %f ms\n", elapsedTime);

//Вычисление времени выполнения при реализации алгоритма линейной алгебры c с использованием библиотеки Thrust:

elapsedTime = saxpyTHRUST();

printf("THRUST Time: %f ms\n", elapsedTime);

//Вычисление времени выполнения при реализации алгоритма линейной алгебры c с использованием библиотеки CuBLAS:

elapsedTime = saxpyCuBLAS();

printf("cuBLAS Time: %f ms\n", elapsedTime);

return 0;

}