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

Кафедра вычислительных систем

Расчетно-графическое задание по дисциплине «Программирование графических процессоров»

Выполнил:
студент группы ИВ-823
Шиндель Э. Д.
ФИО студента

Работу проверил: доцент кафедры вычислительных систем
Милешко А. В.
ФИО преподавателя

Новосибирск 2021 г.

ЗАДАНИЕ

Реализовать функцию умножения треугольной матрицы, хранящейся в “lower” или “upper” виде на вектор, используя cuBLAS, thrust и сырой C Cuda код.

ВЫПОЛНЕНИЕ РАБОТЫ

Задание

Код программ:

CUcode.cu

```
#include <stdio.h>
#include <iostream>
#include <cublas.h>
#include <cublas_v2.h>

#define n 1000
#define INCX 1
#define UPLO CUBLAS_FILL_MODE_UPPER
#define TRANS CUBLAS_OP_N
#define DIAG CUBLAS_DIAG_UNIT

int main()
{
    int matrix_size = (n * n - n) / 2 + n;
    float *vector = new float[n];
    float *result = new float[n];
    float *matrix = new float[matrix_size];
    float *dev_v, *dev_r, *dev_m;
    int lda = n;

    for (int i = 0; i < matrix_size; i++) matrix[i] = i + 1;
    for (int i = 0; i < n; i++) {
        vector[i] = i + 1;
        result[i] = 0.0;
    }

    cudaMalloc(&dev_v, sizeof(float) * n);
    cudaMalloc(&dev_r, sizeof(float) * n);
    cudaMalloc(&dev_m, sizeof(float) * matrix_size);

    cublasHandle_t handle;
    cublasCreate(&handle);
    cublasInit();

    cudaMemcpy(dev_v, vector, sizeof(float) * n, cudaMemcpyHostToDevice);
    cudaMemcpy(dev_r, result, sizeof(float) * n, cudaMemcpyHostToDevice);
    cudaMemcpy(dev_m, matrix, sizeof(float) * matrix_size, cudaMemcpyHostToDevice);

    cublasStrmv(handle, UPLO, TRANS, DIAG, n, dev_m, lda, dev_v, INCX);

    cudaMemcpy(result, dev_r, sizeof(float) * n, cudaMemcpyDeviceToHost);

    cublasShutdown();

    cudaFree(dev_v);
    cudaFree(dev_r);
    cudaFree(dev_m);

    return 0;
}
```

Ccode.cu

```
#include <stdio.h>
#include <cuda.h>

int MAX_THREADS = 256;

__global__ void kernel(int *vector, int *result, int *matrix, int N) {
    int threadId = blockIdx.x * blockDim.x + threadIdx.x;

    int start = (threadId == 0) ? 0 : 2;
    for (int i = 1; i < threadId; i++) start *= 2;
    if (threadId > 1) start++;

    int step = threadId;
    int step_v = threadId;
    int sum = 0;

    for (int i = start; i + step < N; i++) {
        sum += matrix[i + step] * vector[step_v];
        if (i > 1) step++;
        step_v++;
    }
    result[threadId] = sum;
}

int main()
{
    int *vector, *result, *matrix;
    int *dev_v, *dev_r, *dev_m;

    int N = 5000;
    int Matrix_size = (N * N - N) / 2 + N;

    vector = (int *)malloc(sizeof(int) * N);
    result = (int *)malloc(sizeof(int) * N);
    matrix = (int *)malloc(sizeof(int) * Matrix_size);

    for (int i = 0; i < Matrix_size; i++) matrix[i] = i + 1;
    for (int i = 0; i < N; i++) {
        vector[i] = i + 1;
        result[i] = 0;
    }

    cudaMalloc((void **)&dev_v, sizeof(int) * N);
    cudaMalloc((void **)&dev_r, sizeof(int) * N);
    cudaMalloc((void **)&dev_m, sizeof(int) * Matrix_size);

    cudaMemcpy(dev_v, vector, sizeof(int) * N, cudaMemcpyHostToDevice);
    cudaMemcpy(dev_m, matrix, sizeof(int) * N, cudaMemcpyHostToDevice);

    dim3 threadBlock(MAX_THREADS, 1);
    dim3 blockGrid (N / MAX_THREADS + 1, 1, 1);

    kernel<<<blockGrid, threadBlock>>>(dev_v, dev_r, dev_m, N);
    cudaThreadSynchronize();

    cudaMemcpy(result, dev_r, sizeof(int) * N, cudaMemcpyDeviceToHost);

    cudaFree(dev_v);
    cudaFree(dev_r);
    cudaFree(dev_m);

    return 0;
```

```
}
```

THRUSTcode.cu

```
#include <thrust/device_vector.h>
#include <thrust/transform.h>
#include <thrust/inner_product.h>
#include <thrust/execution_policy.h>
#include <thrust/equal.h>
#include <thrust/iterator/constant_iterator.h>
#include <cublas_v2.h>
#include <iostream>
#include <vector>
#include <cstdlib>

using namespace std;

typedef string status_t;
typedef string fillMode_t;

#define n 5000
#define INCX 1

int main()
{
    int Matrix_size = (n * n - n) / 2 + n;
    thrust::device_vector<float> vector(n);
    thrust::device_vector<float> result(n);
    thrust::device_vector<float> matrix(Matrix_size);

    thrust::sequence(vector.begin(), vector.end(), 1, 1);
    thrust::sequence(matrix.begin(), matrix.end(), 1, 1);
    thrust::sequence(result.begin(), result.end(), 0);

    int start = 0;
    int end = n;

    for (int i = 0; i < n; i++) {
        float sum = thrust::reduce(matrix.begin() + start, matrix.begin() + end);
        start = end;
        end = start + (n - 1) - i;
        thrust::fill(result.begin() + i, result.begin() + i + 1, sum);
    }
    thrust::transform(result.begin(), result.end(), vector.begin(), result.begin(),
                                                                    thrust::multiplies<float>());
}
```

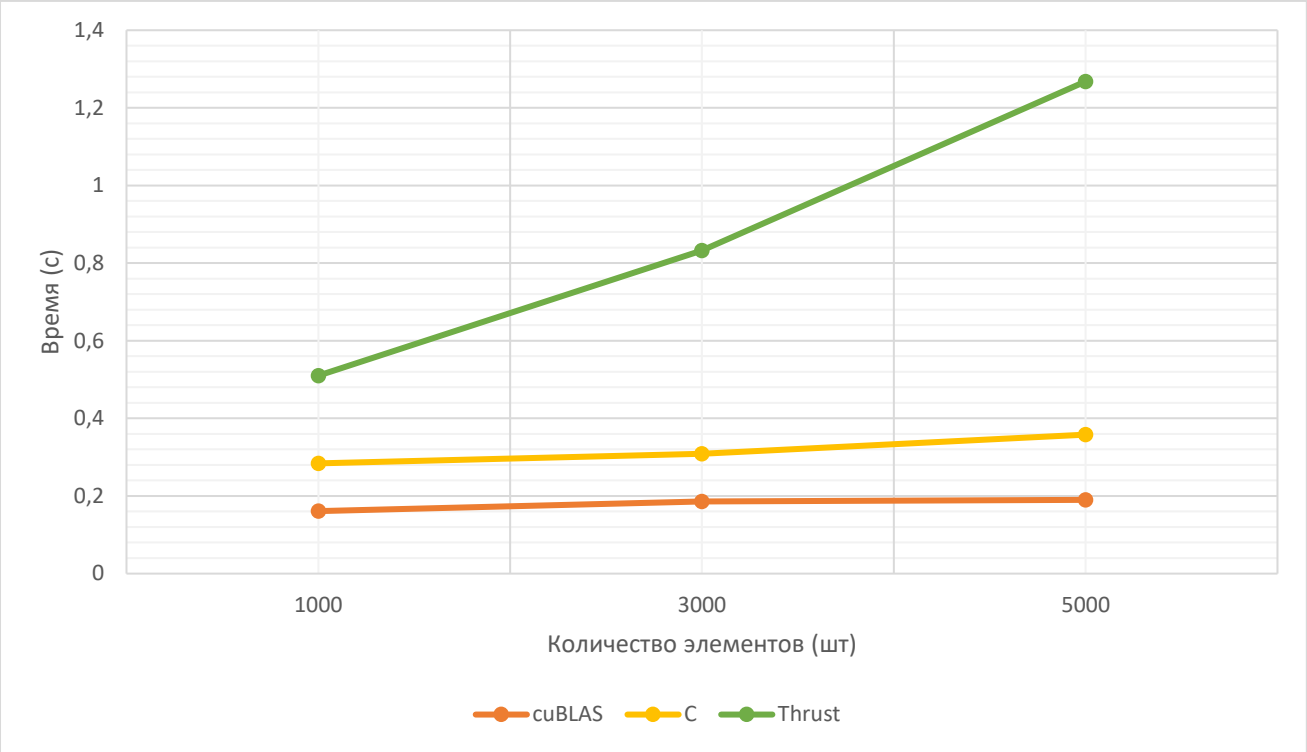

График 1. Зависимость времени выполнения от количества элементов.

```
is-742@linux-47dw:~/Shindel_Lozovoi/Shindel_RGZ> time ./cuBLAS
real    0m0.161s
user    0m0.027s
sys     0m0.110s

is-742@linux-47dw:~/Shindel_Lozovoi/Shindel_RGZ> time ./C
real    0m0.284s
user    0m0.089s
sys     0m0.192s

is-742@linux-47dw:~/Shindel_Lozovoi/Shindel_RGZ> time ./Thrust
real    0m0.510s
user    0m0.153s
sys     0m0.353s
```

Вставка 1. Замеры времени при n = 1000

```
is-742@linux-47dw:~/Shindel_Lozovoi/Shindel_RGZ> time ./cuBLAS
real    0m0.186s
user    0m0.016s
sys     0m0.164s
```

```
is-742@linux-47dw:~/Shindel_Lozovoi/Shindel_RGZ> time ./C
real    0m0.309s
user    0m0.111s
sys     0m0.190s

is-742@linux-47dw:~/Shindel_Lozovoi/Shindel_RGZ> time ./Thrust
real    0m0.832s
user    0m0.140s
sys     0m0.688s
```

Вставка 2. Замеры времени при n = 3000

```
is-742@linux-47dw:~/Shindel_Lozovoi/Shindel_RGZ> time ./cuBLAS
real    0m0.190s
user    0m0.020s
sys     0m0.168s

is-742@linux-47dw:~/Shindel_Lozovoi/Shindel_RGZ> time ./C
real    0m0.358s
user    0m0.094s
sys     0m0.188s

is-742@linux-47dw:~/Shindel_Lozovoi/Shindel_RGZ> time ./Thrust
real    0m1.268s
user    0m0.299s
sys     0m0.962s
```

Вставка 3. Замеры времени при n = 5000