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