МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ

(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

Кафедра вычислительной математики и программирования

**спецкурс «Параллельное программирование»**

**ОТЧЕТ**

**Лабораторная работа № 1**

**«Работа с матрицами»**

Выполнил: Буевич П. И.

Группа: 114-2022, **вариант 3**

Преподаватель: Семенов С. А.

Москва, 2014

**Оглавление**

[1. Постановка задачи](#_heading=h.uyezzsf799mi) **2**

[2. Описание решения](#_heading=h.30j0zll) **2**

[3. Основные моменты кода](#_heading=h.1fob9te) **2**

[4. Результат работы программы](#_heading=h.3znysh7) **3**

[7. Выводы](#_heading=h.3dy6vkm) **4**

[8. Приложения](#_heading=h.1t3h5sf) **4**

# 1. Постановка задачи

Выполнить перемножение двух матриц, одинаковой размерности *N*х*N*.

# 2. Описание решения

В результате умножения двух квадратных матриц *А* и *В* порядка *N*

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

*A=*

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

B=

будет матрица *С* размерностью *N*х*N*:

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

C=

где:

# 3. Основные моменты кода

Так как матрица двумерная, то удобно использовать двумерные блоки и двумерную сетку. Произведем вычисления так, что:

* матрицы расположены в глобальной памяти;
* по одной нити на каждый элемент произведения.

Произведение двух квадратных матриц *A* и *B* размера *N*\**N*, где *N* кратно 16, вычисляется при задании постоянного значения *BLOCK*\_*SIZE* = 16. Реализация для *N* < 16 осуществляется заданием констант *N* и *BLOCK*\_*SIZE* равными значениями:

#define N (4)

#define BLOCK\_SIZE 4

\_\_global\_\_ void mult( float \*a, float \*b, float \*c) {

int bx = blockIdx.x;

int by = blockIdx.y;

int tx = threadIdx.x;

int ty = threadIdx.y;

float sum = 0.0f;

int ia = N \* BLOCK\_SIZE \* by + N \* ty; // смещение для a[i][0]

int ib = BLOCK\_SIZE \* bx + tx; // смещение для b[0][j]

for (int k=0; k < N; k++) // вычисляем элемент

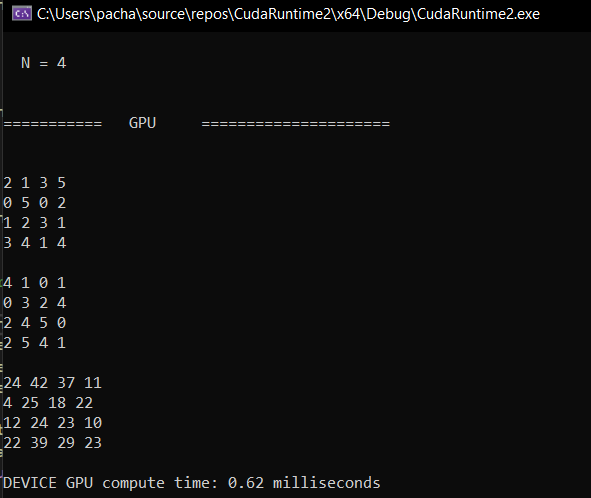
sum += a[ia + k] \* b[ib + k \* N];

int ic = N \* BLOCK\_SIZE \* by + BLOCK\_SIZE \* bx; // смещение для элемента c

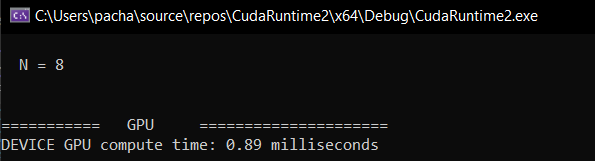
c[ic + N \* ty + tx] = sum; // запоминаем результат

}

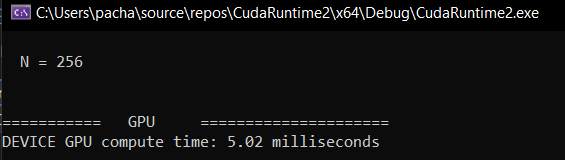
# 4. Результат работы программы



**Рис. 1.** Окно вывода консольного приложения при *N* = 4



**Рис. 2.** Окно вывода консольного приложения при *N* = 8



**Рис. 3.** Окно вывода консольного приложения при *N* = 256

# 7. Выводы

В Лабораторной работе №1 проведен анализ работы различных программ по решению задачи умножения матриц, порядка *N*.

# 8. Приложение

Код программы

#include "cuda\_runtime.h" // CUDA runtime

#include "device\_launch\_parameters.h"

#include <cuda\_profiler\_api.h>

#include <stdio.h>

#include <cstdlib>

#include <ctime>

//#include <vector>

#define N 16

#define BLOCK\_SIZE 16

\_\_global\_\_ void mult( float \*a, float \*b, float \*c) {

int bx = blockIdx.x;

int by = blockIdx.y;

int tx = threadIdx.x;

int ty = threadIdx.y;

float sum = 0.0f;

int ia = N \* BLOCK\_SIZE \* by + N \* ty; // смещение для a[i][0]

int ib = BLOCK\_SIZE \* bx + tx; // смещение для b[0][j]

for (int k=0; k < N; k++) // вычисляем элемент

sum += a[ia + k] \* b[ib + k \* N];

int ic = N \* BLOCK\_SIZE \* by + BLOCK\_SIZE \* bx; // смещение для элемента c

c[ic + N \* ty + tx] = sum; // запоминаем результат

}

int main()

{

printf("\n N = %i\n\n", N );

float \*a,\*b,\*c;

int numBytes = N\*N\*sizeof(float);

a=(float \*)malloc(numBytes);

b=(float \*)malloc(numBytes);

c=(float \*)malloc(numBytes);

//// Тестовый массив 4x4

a[0] = 2.0f; a[1] = 1.0f; a[2] = 3.0f; a[3] = 5.0f;

a[4] = 0.0f; a[5] = 5.0f; a[6] = 0.0f; a[7] = 2.0f;

a[8] = 1.0f; a[9] = 2.0f; a[10] = 3.0f; a[11] = 1.0f;

a[12] = 3.0f; a[13] = 4.0f; a[14] = 1.0f; a[15] = 4.0f;

b[0] = 4.0f; b[1] = 1.0f; b[2] = 0.0f; b[3] = 1.0f;

b[4] = 0.0f; b[5] = 3.0f; b[6] = 2.0f; b[7] = 4.0f;

b[8] = 2.0f; b[9] = 4.0f; b[10] = 5.0f; b[11] = 0.0f;

b[12] = 2.0f; b[13] = 5.0f; b[14] = 4.0f; b[15] = 1.0f;

// Единичные a и b

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

// a[i]=1.0f;

// b[i]=1.0f;

//}

for (int i=0; i<N\*N; i++) c[i]=0.0f;

// создать элементы strat и stop для событий

cudaEvent\_t start, stop;

float gpuTime = 0.0f;

cudaEventCreate ( &start );

cudaEventCreate ( &stop );

cudaEventRecord ( start, 0 );

float \*dev\_a, \*dev\_b, \*dev\_c;

// Выделить память в DRAM

cudaMalloc((void\*\*) &dev\_a, numBytes);

cudaMalloc((void\*\*) &dev\_b, numBytes);

cudaMalloc((void\*\*) &dev\_c, numBytes);

// Скопировать из CPU в DRAM

cudaMemcpy(dev\_a, a,numBytes,cudaMemcpyHostToDevice);

cudaMemcpy(dev\_b, b,numBytes,cudaMemcpyHostToDevice);

cudaMemcpy(dev\_c, c,numBytes,cudaMemcpyHostToDevice);

dim3 threads(BLOCK\_SIZE, BLOCK\_SIZE);

dim3 blocks(N/threads.x, N/threads.y);

// Вызвать ядро

mult<<<blocks, threads>>>(dev\_a,dev\_b,dev\_c);

cudaThreadSynchronize(); // Дождаться окончания расчета

cudaMemcpy(c, dev\_c,numBytes,cudaMemcpyDeviceToHost);

// get data back

cudaEventRecord ( stop, 0 );

// force synchronization

cudaEventSynchronize ( stop );

cudaEventElapsedTime ( &gpuTime, start, stop );

printf("\n=========== GPU =====================\n");

printf("\n");

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

if (i%N==0) printf("\n");

printf("%.0f ",a[i]);

}

printf("\n");

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

if (i%N==0) printf("\n");

printf("%.0f ",b[i]);

}

printf("\n");

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

if (i%N==0) printf("\n");

printf("%.0f ",c[i]);

}

printf("\n\n");

// print the GPU times

printf("DEVICE GPU compute time: %.2f milliseconds\n\n", gpuTime );

cudaFree( dev\_a);

cudaFree( dev\_b);

cudaFree( dev\_c);

free(a);

free(b);

free(c);

system("pause");

cudaProfilerStop();

return 0;

}