

МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

Кафедра вычислительной математики и программирования

спецкурс «Параллельные и распределенные вычисления»

ОТЧЕТ

Лабораторная работа № 1
«Освоение программного обеспечения среды программирования
NVIDIA»

Выполнил: Патрикеева Л.В.

Группа: М8О-114М-22, вариант 3

Преподаватель: Семенов С. А.

Москва, 2022

Содержание

1. Постановка задачи.....	2
2. Описание решения	2
3. Аппаратное обеспечение и ПО	2
4. Основные моменты кода	3
5. Результат работы программы	4
6. Сравнение скорости выполнения на CPU и GPU	6
7. Выводы.....	6
8. Приложения	7

1. Постановка задачи

Поиск A^{-1} для заданной матрицы.

2. Описание решения

Для решения данной задачи будет реализован метод Гаусса с выбором главного элемента по столбцу. Саму матрицу, которая будет подаваться на вход, будем хранить по столбцам в линеаризованном виде, т.е. в одномерном массиве. После считывания заданной матрицы записываем единичную матрицу. По результату применения метода Гаусса на месте единичной матрицы получим обратную матрицу. Поиск максимального элемента в столбце будем реализовывать при помощи библиотеки Thrust. Прямой и обратные ходы будем выполнять на GPU.

3. Аппаратное обеспечение и ПО

Название: GeForce 840M

Размер глобальной памяти: 3150381056

Размер константной памяти: 65536

Размер разделяемой памяти: 49152

Регистров на блок: 32768

Максимум потоков на блок: 1024

Количество мультипроцессоров: 3

RAM: 16 GB

SSD: 500 GB

OS: Windows 10
IDE: Visual Studio 2017

4. Основные моменты кода

Для того чтобы использовать поиск максимума при помощи библиотеки Thrust нужен компаратор, который будет правильно сравнивать объекты.

```
struct comparator {  
    __host__ __device__ bool operator()(double a, double b) {  
        return abs(a) < abs(b);  
    }  
};
```

Сохраняем элементы матрицы в линеаризованном виде:

```
for (int i = 0; i < n; i++) {  
    for (int j = 0; j < n; j++) {  
        scanf("%lf", &matrix[j * n + i]);  
    }  
}
```

Была создана отдельная функция, которая меняет столбцы местами:

```
__global__ void change(double *mat, int n, int ind, int max_r) {  
    int idx = blockDim.x * blockIdx.x + threadIdx.x;  
    int offsetx = gridDim.x * blockDim.x;  
    for (int i = idx + ind; i < 2 * n; i += offsetx)  
    {  
        double c = mat[ind + n * i];  
        mat[ind + n * i] = mat[n * i + max_r];  
        mat[n * i + max_r] = c;  
    }  
}
```

Основной обработкой в данном задании занимаются global функция gauss1 и gauss2. Выполняется прямой и обратный ход.

```
__global__ void gauss1(double *mat, int n, int ind) {  
    int idx = blockDim.x * blockIdx.x + threadIdx.x;  
    int idy = blockDim.y * blockIdx.y + threadIdx.y;  
    int offsetx = blockDim.x * gridDim.x;  
    int offsety = blockDim.y * gridDim.y;  
    for (int i = idx + ind + 1; i < n; i += offsetx) {  
        for (int j = idy + ind + 1; j < 2 * n; j += offsety) {  
            mat[j * n + i] += mat[j * n + ind] * (-mat[ind * n + i] /  
            mat[ind * n + ind]);  
        }  
    }  
}
```

```
}}
```

```
__global__ void gauss2(double *mat, int n, int ind) {  
    int idx = blockDim.x * blockIdx.x + threadIdx.x;  
    int idy = blockDim.y * blockIdx.y + threadIdx.y;  
    int offsetx = blockDim.x * gridDim.x;  
    int offsety = blockDim.y * gridDim.y;  
    for (int i = ind - 1 - idx; i >= 0; i -= offsetx) {  
        for (int j = idy + n; j < 2 * n; j += offsety) {  
            mat[j * n + i] += mat[j * n + ind] *  
            (-mat[ind * n + i] / mat[ind * n + ind]);  
        }  
    }  
}
```

Все стандартные функции возвращают код исполнения, поэтому они обёрнуты в макрос, который проверяет исполнение на ошибки.

5. Результат работы программы

Вывод обратной матрицы для заданной с клавиатуры матрицы размером $N \times N$

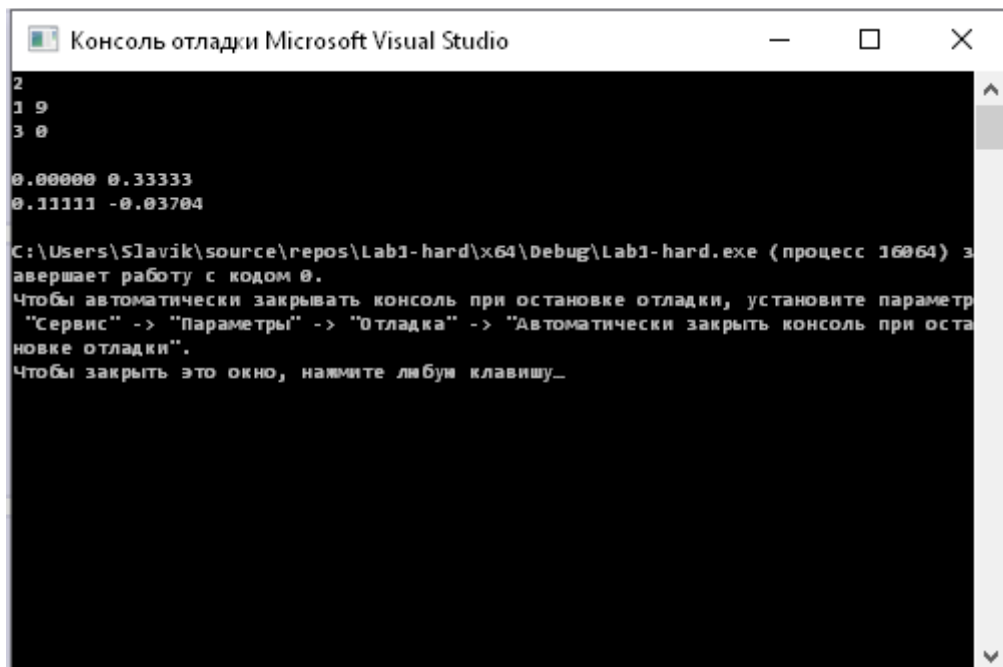
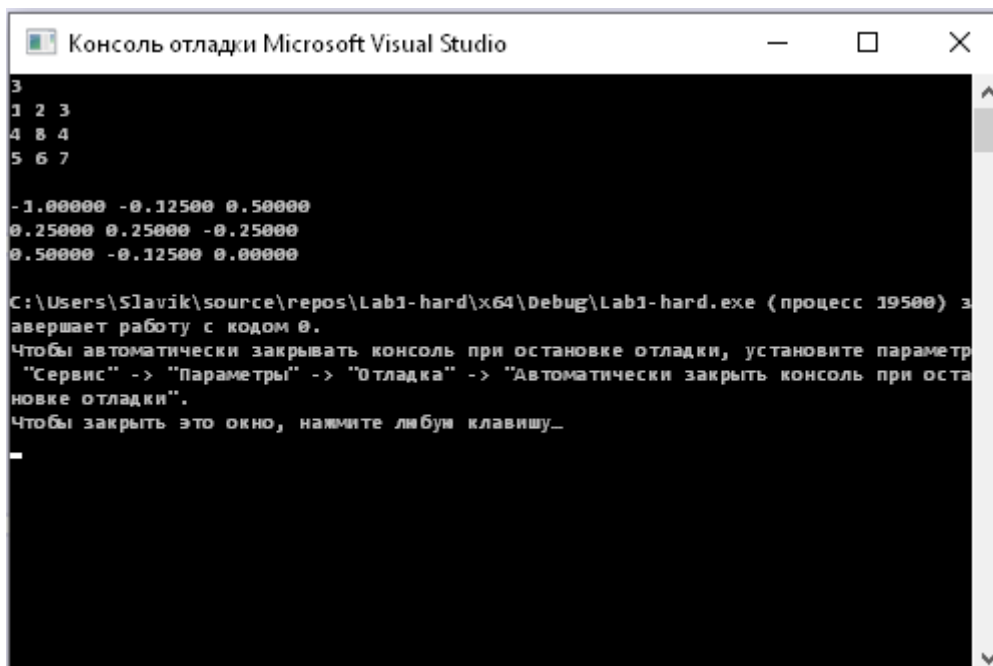


Рис. 1. Окно вывода консольного приложения при $N = 2$



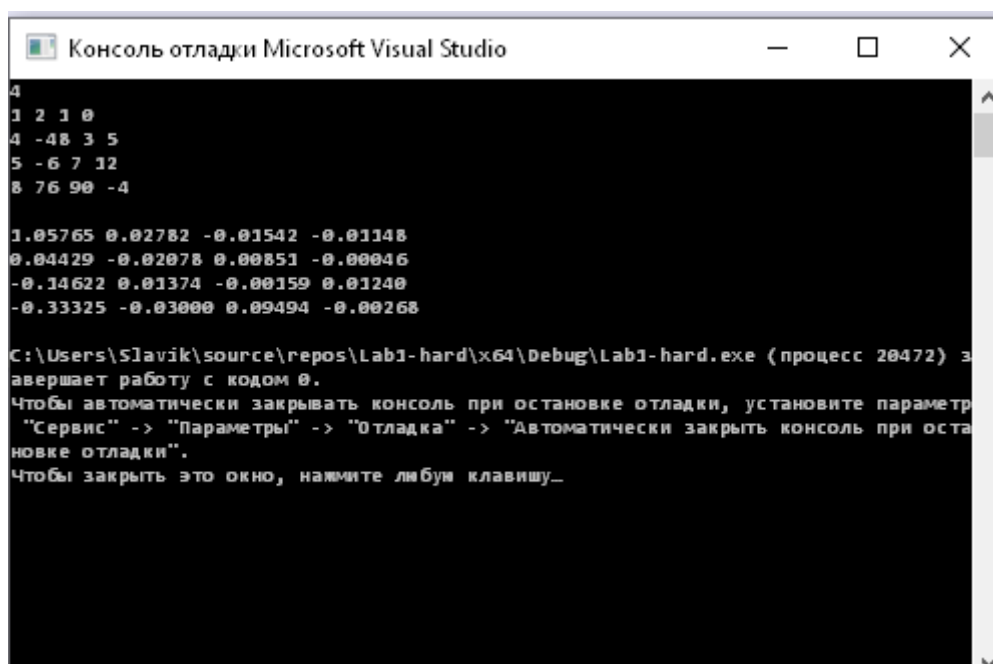
Консоль отладки Microsoft Visual Studio

```
3
1 2 3
4 8 4
5 6 7

-1.00000 -0.12500 0.50000
0.25000 0.25000 -0.25000
0.50000 -0.12500 0.00000

C:\Users\Slavik\source\repos\Lab1-hard\x64\Debug\Lab1-hard.exe (процесс 19500) з
авершает работу с кодом 0.
Чтобы автоматически закрывать консоль при остановке отладки, установите параметр
"Сервис" -> "Параметры" -> "Отладка" -> "Автоматически закрыть консоль при оста
новке отладки".
Чтобы закрыть это окно, нажмите любую клавишу_
```

Рис. 2. Окно вывода консольного приложения при $N = 3$



Консоль отладки Microsoft Visual Studio

```
4
1 2 1 0
4 -48 3 5
5 -6 7 12
8 76 90 -4

1.05765 0.02782 -0.01542 -0.01148
0.04429 -0.02078 0.00851 -0.00046
-0.14622 0.01374 -0.00159 0.01240
-0.33325 -0.03000 0.09494 -0.00268

C:\Users\Slavik\source\repos\Lab1-hard\x64\Debug\Lab1-hard.exe (процесс 20472) з
авершает работу с кодом 0.
Чтобы автоматически закрывать консоль при остановке отладки, установите параметр
"Сервис" -> "Параметры" -> "Отладка" -> "Автоматически закрыть консоль при оста
новке отладки".
Чтобы закрыть это окно, нажмите любую клавишу_
```

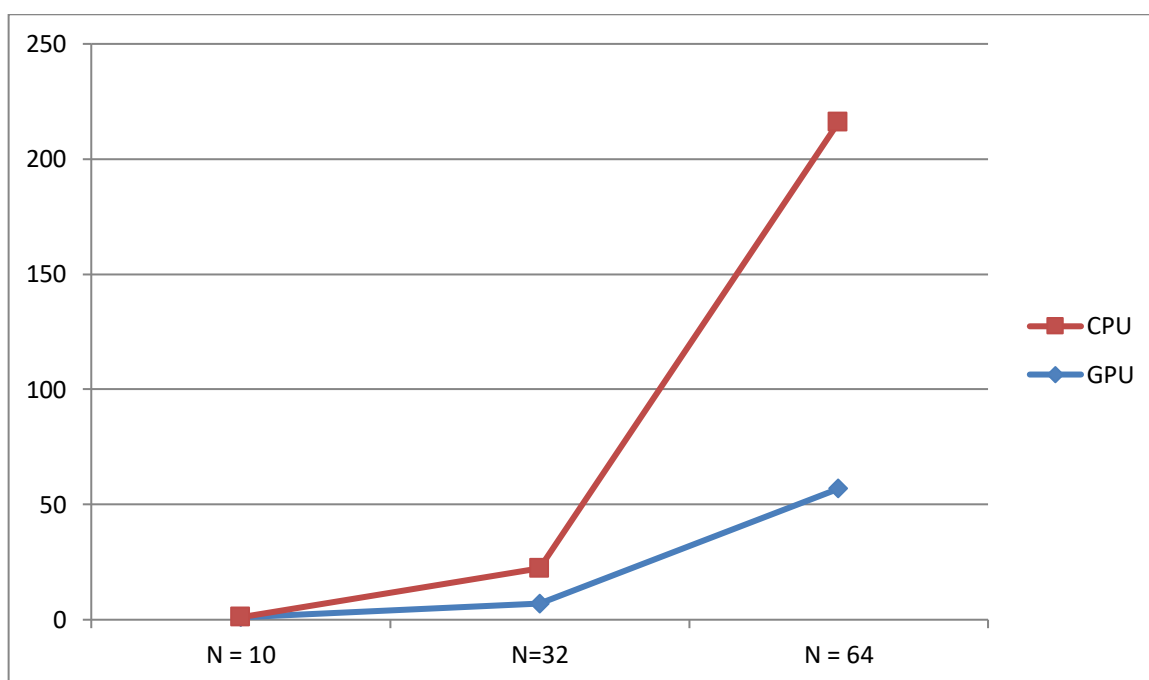
Рис. 3. Окно вывода консольного приложения при $N = 4$

6. Сравнение скорости выполнения на CPU и GPU

Замеры времени работы (в секундах) ядер с различными конфигурациями и различными входными данными, а так же сравнение с CPU:

	N = 10	N = 32	N = 64
<16, 16>	1.05	7.001	56.87
<32,16>	1.12	6.82	56.27
<32,32>	2.2998	10.07	58.703
CPU	0.01001	15.23	159.13

N – размер матрицы, N x N – количество элементов в матрице



7. Выводы

При выполнении работы был реализован метод Гаусса на GPU. Благодаря распараллеливанию, работа алгоритма с большими матрицами на GPU происходит намного быстрее, чем на CPU, начиная с $N \sim 30$. При маленьких значениях $N \sim 10$ программа на CPU работает во много раз быстрее. Чем больше становится размер матрицы, тем сильнее выигрыш по времени у GPU по сравнению с CPU.

8. Приложения

Код программы

Ссылка на код в github:

<https://github.com/patrikeyeva/CUDA-Labs/tree/main/Lab1-hard>

```
#include "stdio.h"
#include "stdlib.h"
#include "math.h"
#include <iostream>
#include <cstdlib>
#include <string>
#include <vector>
#include <thrust/extrema.h>
#include <thrust/device_vector.h>

using namespace std;

#define CSC(call)

do {

    cudaError_t res = call;

    if (res != cudaSuccess) {
        fprintf(stderr, "ERROR in %s:%d. Message: %s\n",
                __FILE__, __LINE__, cudaGetErrorString(res));
        exit(0);
    }

} while (0)

__global__ void change(double *mat, int n, int ind, int max_r) {
    int idx = blockDim.x * blockIdx.x + threadIdx.x; // Абсолютный номер потока
    int offsetx = gridDim.x * blockDim.x; // Общее кол-во потоков

    for (int i = idx + ind; i < 2 * n; i += offsetx)
    {
        double c = mat[ind + n * i];
        mat[ind + n * i] = mat[n * i + max_r];
        mat[n * i + max_r] = c;
    }
}

__global__ void gauss1(double *mat, int n, int ind) {
    int idx = blockDim.x * blockIdx.x + threadIdx.x; ///// Абсолютный номер потока
    int idy = blockDim.y * blockIdx.y + threadIdx.y;
    int offsetx = blockDim.x * gridDim.x; // Общее кол-во потоков
    int offsety = blockDim.y * gridDim.y;
```

```

        for (int i = idx + ind + 1; i < n; i += offsetx) {
            for (int j = idy + ind + 1; j < 2 * n; j += offsety) {
                mat[j * n + i] += mat[j * n + ind] * (-mat[ind * n + i] / mat[ind * n
+ ind]);
            }
        }
    }
}

__global__ void gauss2(double *mat, int n, int ind) {

    int idx = blockDim.x * blockIdx.x + threadIdx.x;  ///// Абсолютный номер потока
    int idy = blockDim.y * blockIdx.y + threadIdx.y;
    int offsetx = blockDim.x * gridDim.x;  // Общее кол-во потоков
    int offsety = blockDim.y * gridDim.y;

    for (int i = ind - 1 - idx; i >= 0; i -= offsetx) {
        for (int j = idy + n; j < 2 * n; j += offsety) {
            mat[j * n + i] += mat[j * n + ind] * (-mat[ind * n + i] / mat[ind * n
+ ind]);
        }
    }
}

struct comparator {
    __host__ __device__ bool operator()(double a, double b) {          // Функция
        // которая сравнивает объекты на "<"
        return abs(a) < abs(b);
        // operator() - переопределение оператора "(" для экземпляра этой структуры
    }
};

int main() {

    int n; //размер матрицы
    scanf("%d", &n);
    double *matrix = (double*)malloc(sizeof(double) * 2 * n * n);
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            scanf("%lf", &matrix[j * n + i]);
        }
    }
    printf("\n");

    // добавляем единичную
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            if (i == j) {
                matrix[i * n + j + n * n] = 1.0;
            }
            else {
                matrix[i * n + j + n * n] = 0.0;
            }
        }
    }

    double *dev_matrix;
    cudaMalloc(&dev_matrix, sizeof(double) * 2 * n * n);
    cudaMemcpy(dev_matrix, matrix, sizeof(double) * 2 * n * n,
cudaMemcpyHostToDevice);
}

```



```

    comparator comp;
    thrust::device_ptr<double> ptr = thrust::device_pointer_cast(dev_matrix);

    for (int i = 0; i < n - 1; i++) {
        thrust::device_ptr<double> p_max = thrust::max_element(ptr + i + n * i, ptr
+ n * (i + 1), comp);
        int max_row = p_max - ptr - n * i;

        if (max_row != i)
        {
            change << <256, 256 >> > (dev_matrix, n, i, max_row);
            CSC(cudaGetLastError());
        }
        gauss1 << < dim3(32, 16), dim3(32, 16) >> > (dev_matrix, n, i);
        CSC(cudaGetLastError());
    }

    for (int i = n - 1; i > 0; i--) {
        gauss2 << < dim3(32, 16), dim3(32, 16) >> > (dev_matrix, n, i);
        CSC(cudaGetLastError());
    }

    cudaMemcpy(matrix, dev_matrix, sizeof(double) * 2 * n * n,
cudaMemcpyDeviceToHost);
    cudaFree(dev_matrix);
    CSC(cudaGetLastError());

    for (int i = 0; i < n; i++) {
        for (int j = n; j < 2 * n; j++) {
            matrix[j * n + i] /= matrix[i * n + i];
        }
    }

    for (int i = 0; i < n; ++i) {
        for (int j = n; j < 2 * n; ++j) {
            printf("%.51f ", matrix[j * n + i]);

        }
        printf("\n");
    }

    free(matrix);
    return 0;
}

```