Министерство цифрового развития, связи
и массовых коммуникаций Российской Федерации

Сибирский государственный университет
телекоммуникаций и информатики

Кафедра прикладной математики и кибернетики

ЛАБОРАТОРНАЯ РАБОТА №2

По дисциплине: «Программирование графических процессоров»

Выполнили:

Студенты 3 курса группы ИП-111
Корнилов А.А.,
Попов М.И.,
Толкач А.А.

Проверил:

Профессор кафедры ПМиК
Малков Е.А.

Новосибирск, 2024

Задание: 1. Определить при какой длине векторов имеет смысл распараллеливать операцию сложения, используя потоки CPU или GPU.

2. Определить оптимальное количество потоков POSIX для распараллеливания.

3. Определить зависимость времени выполнения операции сложения на GPU от длины векторов (выбирать количество нитей равным длине вектора).

Цель: начальное знакомство с распараллеливанием кода на GPU .

Выполнение работы:

Для первого задания была написана простоя программа для сложения двух векторов используя GPU, количество векторов задано n = 10000

```
#include "cuda_runtime.h"
#include "device_launch_parameters.h"
#include <stdio.h>
#include <iostream>

__global__ void vectorAdd(const float* a, const float* b, float* c, int n) {
    int i = blockIdx.x * blockDim.x + threadIdx.x;
    if (i < n) {
        c[i] = a[i] + b[i];
    }
}

int main() {
    int n = 10000;
    float elapsedTime;
    cudaEvent_t start, stop;

    float* d_a, * d_b, * d_c;
    cudaMalloc((void**)&d_a, n * sizeof(float));
    cudaMalloc((void**)&d_b, n * sizeof(float));
    cudaMalloc((void**)&d_c, n * sizeof(float));

    float* h_a = new float[n];
    float* h_b = new float[n];
    for (int i = 0; i < n; ++i) {
        h_a[i] = i;
        h_b[i] = i * 2;
    }

    cudaMemcpy(d_a, h_a, n * sizeof(float), cudaMemcpyHostToDevice);
    cudaMemcpy(d_b, h_b, n * sizeof(float), cudaMemcpyHostToDevice);


    // Вычисляем количество блоков и нитей на блок
    int blockSize = 8192;
    int numBlocks = 8192;

    //int numBlocks = (n + blockSize - 1) / blockSize;
    cudaEventCreate(&start);
    cudaEventCreate(&stop);

    cudaEventRecord(start, 0);
```

```
    vectorAdd << < numBlocks, blockSize >> > (d_a, d_b, d_c, n);

    cudaEventRecord(stop, 0);
    cudaEventSynchronize(stop);
    cudaEventElapsedTime(&elapsedTime, start, stop);

    float* h_c = new float[n];
    cudaMemcpy(h_c, d_c, n * sizeof(float), cudaMemcpyDeviceToHost);

    // Выводим результат
//    std::cout << "Result: ";
//    for (int i = 0; i < n; ++i) {
//        std::cout << h_c[i] << " ";
//    }
//    std::cout << std::endl;
    std::cout << elapsedTime << std::endl;


    delete[] h_a;
    delete[] h_b;
    delete[] h_c;
    cudaFree(d_a);
    cudaFree(d_b);
    cudaFree(d_c);

    return 0;
}
```

Листинг 1 – программа lab01_1G_2.cu

Команда компиляции и результат работы программы:

```
PS D:\Projects\CUDA\2> nvcc .\lab01_1G_2.cu -o lab01_1G_2
lab01_1G_2.cu
tmpxft_00004e68_00000000-10_lab01_1G_2.cudafe1.cpp
   Создается библиотека lab01_1G_2.lib и объект lab01_1G_2.exp
PS D:\Projects\CUDA\2> .\lab01_1G_2
0.799744
```

Программа создает два вектора которые значение которых заполнены I и i*2 и складывает их, используя CUDA технологию. Время работы с 8192 нитью и потоками составило 0.799744

Далее была написана программа с такой же целью которая использует возможности процессора

```
#include <iostream>
#include <vector>
#include <thread>
```

```
#include <time.h>

void vectorAdd(const std::vector<float> &a, const std::vector<float> &b,
std::vector<float> &c, int start, int end) {
    for (int i = start; i < end; ++i) {
        c[i] = a[i] + b[i];
    }
}

int main() {
    int n = 10000;
    std::vector<float> a(n), b(n), c(n);
    for (int i = 0; i < n; ++i) {
        a[i] = i;
        b[i] = i * 2;
    }
    //количество поток создаться исходя из потоков процессора
    int numThreads = std::thread::hardware_concurrency();
    std::vector<std::thread> threads;
    for (int i = 0; i < numThreads; ++i) {
        int start = i * (n / numThreads);
        int end = (i == numThreads - 1) ? n : (i + 1) * (n / numThreads);
        threads.emplace_back(vectorAdd, std::ref(a), std::ref(b), std::ref(c), start,
end);
    }

    for (auto &thread : threads) {
        thread.join();
    }

    // Выводим результат
//    std::cout << "Result: ";
//    for (int i = 0; i < n; ++i) {
//        std::cout << c[i] << " ";
//    }
//    std::cout << std::endl;

    return 0;
}
```

Листинг 2 – программа lab11_2.cpp

Команда компиляции и результат работы программы:

```
miron@DESKTOP-UMC1Q46:/mnt/d/Projects/CUDA/2$ g++ lab01_1C.cpp -o
lab01_1C
miron@DESKTOP-UMC1Q46:/mnt/d/Projects/CUDA/2$ time(./lab01_1C)

real    0m0,008s
user    0m0,004s
sys     0m0,000s
```

Программа берет количество потоков исходя из потоков процессора и складывает вектора используя threads, время работы 0,008 секунд

Экспериментируя с количеством вычислительных блоков и количеством нитей делая выборку с значениями 256,512,1024,2048,4096,8192 по 5 раз получил значения:

256	256	512	1024	2048	4096	8192
1	0,4608	0,44544	0,402272	0,47104	0,380928	0,374784
2	0,532352	0,4096	0,425984	0,401408	0,39936	0,372576
3	0,7504	0,417792	0,402432	0,365568	0,385984	0,380928
4	0,38912	0,392192	0,4096	0,397312	0,404352	0,425984
5	0,425984	0,423936	0,408416	0,403552	0,382976	0,377664

Таблица 1 – выборка с количеством нитей 256

512	256	512	1024	2048	4096	8192
1	0,467968	0,412672	0,39424	0,407616	0,545792	0,39936
2	0,396288	0,390144	0,425984	0,367392	0,388096	0,374976
3	0,400384	0,398336	0,56832	0,601088	0,418816	0,391008
4	0,397312	0,429056	0,410624	0,421696	0,663552	0,380064
5	0,393216	0,407552	0,44032	0,499904	0,403456	0,444416

Таблица 2 – выборка с количеством нитей 512

1024	256	512	1024	2048	4096	8192
1	0,427008	0,413696	0,433984	0,459776	0,40976	0,421888
2	0,377664	0,430976	0,415744	0,378816	0,413696	0,363488
3	0,4096	0,410496	0,422912	0,410624	0,407552	0,545792
4	0,99328	0,42496	0,415744	0,374784	0,401408	0,373792
5	0,39936	0,413696	0,405504	0,402464	0,40448	0,38912

Таблица 3 – выборка с количеством нитей 1024

2048	256	512	1024	2048	4096	8192
1	0,484288	0,817152	0,500576	0,480416	0,41984	0,418816
2	0,406368	0,408576	0,420864	0,366784	0,452608	0,365568
3	0,398336	0,410624	0,443392	0,36864	0,40672	0,406528
4	0,414688	0,44032	0,423936	0,467968	0,403456	0,38688
5	0,951296	0,39424	0,423936	0,366592	0,40768	0,371712

Таблица 4 – выборка с количеством нитей 2048

4096	256	512	1024	2048	4096	8192
1	0,458752	0,458752	0,51712	0,388992	0,411648	0,43008
2	0,428032	0,4792	0,46992	0,402432	0,36864	0,420736
3	0,408576	0,417696	0,504832	0,379904	0,387072	0,376928
4	0,422912	0,449536	0,472064	0,398336	0,458752	0,47104
5	0,410624	0,567296	0,469888	0,361696	0,53248	0,393216

Таблица 5 – выборка с количеством нитей 4096

8192	256	512	1024	2048	4096	8192
1	0,477184	0,490496	0,612352	0,400288	0,717824	0,57152
2	0,420864	0,452608	0,589824	0,41984	0,379904	0,39952
3	0,422912	0,47616	0,57856	0,418816	0,39936	0,38608
4	0,497664	0,462848	0,581632	0,39936	0,37376	0,4096
5	0,467968	0,474976	0,718848	0,42384	0,41168	0,365568

Таблица 6 – выборка с количеством нитей 8192

	256	512	1024	2048	4096	8192	blockSize
256	0,511731	0,417792	0,409741	0,407776	0,39072	0,386387	
512	0,411034	0,407552	0,447898	0,459539	0,483942	0,397965	
1024	0,521382	0,418765	0,418778	0,405293	0,407379	0,418816	
2048	0,530995	0,494182	0,442541	0,41008	0,418061	0,389901	
4096	0,425779	0,474496	0,486765	0,386272	0,431718	0,4184	
8192	0,457318	0,471418	0,616243	0,412429	0,456506	0,426458	
numBlocks							

Таблица 7 – таблица средних значений от каждой выборки по нитям

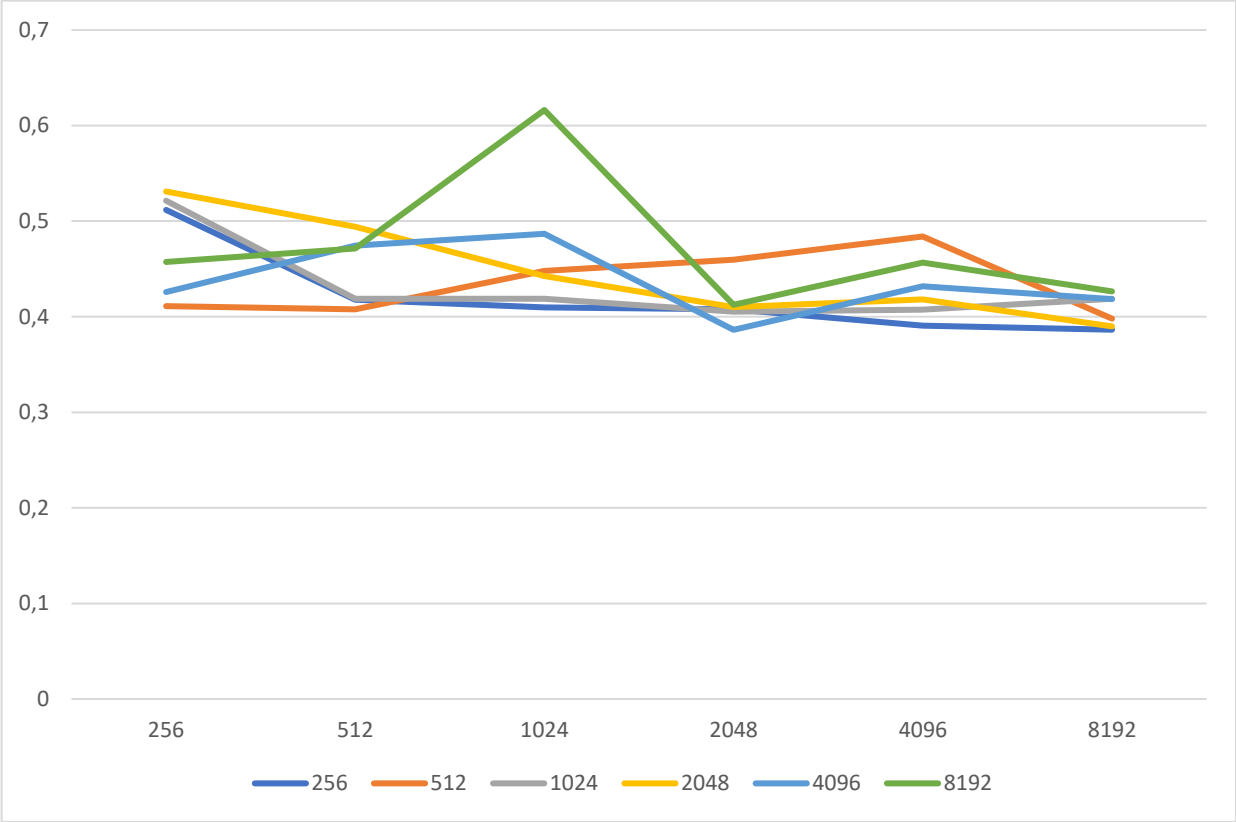

График 1 – График средних значений по нитям от каждого размера блока

В результате исследований получилось что самое низкое среднее значение времени вычисление оказалось при размере блока 2048 и размером нити 4096 = 0,386272.

Самое большое время вычисление получилось при размере блока 1024 и размере нити 8192 = 0,6162432.

При значении нити равной длиннее вектора получилось:

N	256	512	1024	2048	4096	8192
1	0,494432	0,516096	0,626688	0,397248	0,421888	0,381952
2	0,456704	0,532384	0,635904	0,380928	0,559104	0,407328
3	1,00544	0,498688	0,7136	0,370688	0,377856	0,375776
4	0,459776	0,503808	0,615392	0,43008	0,379904	0,419808
5	0,45568	0,536576	0,630592	0,397312	0,406528	0,436224
СРЗ	0,574406	0,51751	0,644435	0,395251	0,429056	0,404218

Таблица 8 – таблица средних значений от нити равной n

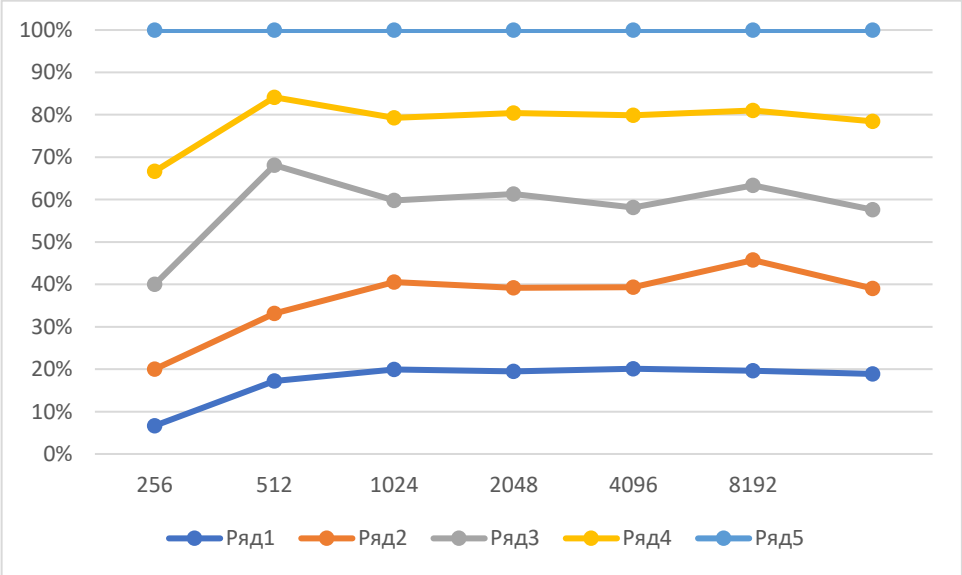

График 2 – График средних значений по нитям от n

Минимальное значение получилось при количество блоков 2048 = 0,3952512, максимальное при 1024 = 0,6444352

Таблица Excel будет приведена в том-же письме что и отчет, вычисления проводились на GTX 1050ti 4Gb