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-211
Оганесян А.С.
Лацук А.Ю.

Проверил:

Профессор кафедры ПМиК
Малков Е.А.

Новосибирск 2025

**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 были написаны программы для сложения n векторов для одного потока, нескольких потоков CPU и с использованием GPU.

**Ход выполненеия работы:**

1. Напишем программу для сложения векторов длинами от 10 до ста миллионов, без использования многопоточности и CUDA и измерим время

```
#include <iostream>
#include <vector>
#include <thread>
#include <chrono>
using namespace std;
typedef std::chrono::milliseconds ms;
typedef std::chrono::nanoseconds ns;

int main()
{
  for (long n = 10; n <= 100000000; n *=10){
    cout << endl << "n = " << n << endl;
    vector<float> a(n), b(n), c(n);
  chrono::time_point<chrono::system_clock> start, end;

  for (int i = 0; i < n; ++i)
  {
    a[i] = i;
    b[i] = i * 2;
  }

  start = chrono::system_clock::now();
  for (int i = 0; i < n; ++i)
  {
    c[i] = a[i] + b[i];
```

```
  }
  end = chrono::system_clock::now();

  cout << "Wasted time: " << chrono::duration_cast<ms>(end -
start).count() << "ms" << endl
       << chrono::duration_cast<ns>(end - start).count() << "ns"
<< endl;
}
  return 0;
}
```

Листинг 1 – программа main.cpp

Результат работы программы:

```
n = 10
Wasted time: 0ms
127ns

n = 100
Wasted time: 0ms
390ns

n = 1000
Wasted time: 0ms
3561ns

n = 10000
Wasted time: 0ms
34193ns

n = 100000
Wasted time: 0ms
423438ns

n = 1000000
Wasted time: 4ms
4131491ns
```

```
n = 10000000
Wasted time: 37ms
37907812ns

n = 100000000
Wasted time: 378ms
378732826ns
```

2. Напишем программу для сложения векторов длинами от 10 до ста миллионов, с использованием 12 потоков и измерим время сложения

```
#include <iostream>
#include <vector>
#include <thread>
#include <chrono>
using namespace std;

typedef std::chrono::milliseconds ms;
typedef std::chrono::nanoseconds ns;

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

int main()
{
  for (long n = 10; n <= 100000000; n *= 10)
  {
    cout << endl
         << "n = " << n << endl;
    vector<float> a(n), b(n), c(n);
    int numThreads = thread::hardware_concurrency();
    // int numThreads = 4;

    chrono::time_point<chrono::system_clock> start, end;

    for (int i = 0; i < n; ++i)
    {
      a[i] = i;
```

```
      b[i] = i * 2;
    }

    vector<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, ref(a), ref(b), ref(c),
start, end);
    }

    start = chrono::system_clock::now();
    for (auto &thread : threads)
    {
      thread.join();
    }
    end = chrono::system_clock::now();

    cout << "Wasted time: " << chrono::duration_cast<ms>(end -
start).count() << "ms" << endl
         << chrono::duration_cast<ns>(end - start).count() << "ns"
<< endl;
  }
}
```

Листинг 2 – программа main_2.cpp

Результат работы программы:

```
n = 10
Wasted time: 0ms
52478ns

n = 100
Wasted time: 0ms
42404ns

n = 1000
Wasted time: 0ms
```

```
41151ns

n = 10000
Wasted time: 0ms
65896ns

n = 100000
Wasted time: 0ms
79459ns

n = 1000000
Wasted time: 1ms
1212913ns

n = 10000000
Wasted time: 9ms
9028519ns

n = 100000000
Wasted time: 86ms
86665454ns
```

Листинг 3 – программа main.cu

3. Напишем программу для сложения векторов длинами от 10 до ста миллионов, с использованием CUDA и измерим время сложения

```
#include "cuda_runtime.h"
#include "device_launch_parameters.h"
#include <stdio.h>
#include <iostream>
#include <chrono>
using namespace std;
typedef std::chrono::milliseconds ms;
typedef std::chrono::nanoseconds ns;

__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() {
    for (long n = 10; n <= 100000000; n *= 10)
  {
    cout << endl
         << "n = " << n << endl;
    float elapsedTime;
    cudaEvent_t start, stop;
    chrono::time_point<chrono::system_clock> start_chrono,
end_chrono;

    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 = 1024;
    int numBlocks = n;

    cudaEventCreate(&start);
    cudaEventCreate(&stop);

    cudaEventRecord(start, 0);
    start_chrono = chrono::system_clock::now();
    vectorAdd <<< numBlocks, blockSize >>> (d_a, d_b, d_c, n);
    cudaEventRecord(stop, 0);
    end_chrono = chrono::system_clock::now();

    cudaEventSynchronize(stop);
```

```
    cudaEventElapsedTime(&elapsedTime, start, stop);

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

    cout <<"CUDA Event time: "<< elapsedTime * 1000 << "ns" <<
endl
         <<"Chrono time: "<< chrono::duration_cast<ms>(end_chrono
- start_chrono).count() << "ms"
        << endl << chrono::duration_cast<ns>(end_chrono -
start_chrono).count() << "ns" << endl;


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

  }
}
```

Результат работы программы:

```
n = 10
CUDA Event time: 155.52ns
Chrono time: 0ms
151498ns

n = 100
CUDA Event time: 8.192ns
Chrono time: 0ms
10385ns

n = 1000
CUDA Event time: 21.504ns
Chrono time: 0ms
8642ns
```

```
n = 10000
CUDA Event time: 189.44ns
Chrono time: 0ms
27905ns

n = 100000
CUDA Event time: 1397.28ns
Chrono time: 0ms
14626ns

n = 1000000
CUDA Event time: 13689.5ns
Chrono time: 0ms
29096ns

n = 10000000
CUDA Event time: 13689.5ns
Chrono time: 0ms
30341ns

n = 100000000
CUDA Event time: 13689.5ns
Chrono time: 0ms
```

Измерим время работы программ на разном векторов:

| размер вектора | один поток(ns) | 12 потоков(ns) | CUDA(ns) |
|---|---|---|---|
| 10 | 127 | 52478 | 151498 |
| 100 | 390 | 42404 | 10385 |
| 1000 | 3561 | 41151 | 8642 |
| 10000 | 34193 | 65896 | 27905 |
| 100000 | 423438 | 79459 | 14626 |
| 1000000 | 4131491 | 121913 | 29096 |
| 10000000 | 379007812 | 9028519 | 30341 |
| 100000000 | 378732826 | 86665454 | 13689 |

Таблица 1 – Замеры программ с разным размером векторов.

По таблице видно что использовать один поток эффективнее на векторах размером менее 100000, потом эффективней использовать вычисления с использованием нескольких потоков. GPU же показывает наибольшую эффективность на векторах размером больше 1000.

Вывод: Мы познакомились с распараллеливанием кода на GPU и определили, что она наиболее эффективен на большом количестве данных