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.

1. Определить оптимальное количество потоков POSIX для распараллеливания.
2. 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:

Таблица 1 – Замеры программ с разным количеством 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