Министерство цифрового развития, связи и массовых коммуникаций Российской Федерации

Сибирский государственный университет телекоммуникаций и информатики

Кафедра прикладной математики и кибернетики

# ЛАБОРАТОРНАЯ РАБОТА №4

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. Напишите программу сложения двух векторов на GPU.

2.Зафиксируйте длину векторов равной 1<<20.

3. Проведите серию запусков программы варьируя количество нитей в

блоке, 1, 16, 32, 64,128, …, 1024, для запуска ядра.

3. Определите зависимость времени выполнения ядра, вычисляющего

сумму векторов, от конфигурации нитей. Используйте для

определения времени события CUDA и профилировщики.

**Цель:** априорное понимание влияния конфигурации нитей на

производительность выполнения кода на GPU.

1. Напишем программу для сложения векторов и определением времени при помощи времени событий CUDA:

| #include <stdio.h> #include <cuda\_runtime.h>  #define N (1 << 20)  \_\_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() {  float \*h\_a, \*h\_b, \*h\_c;  float \*d\_a, \*d\_b, \*d\_c;  size\_t size = N \* sizeof(float);   // Выделение памяти на хосте  h\_a = (float\*)malloc(size);  h\_b = (float\*)malloc(size);  h\_c = (float\*)malloc(size);   // Инициализация векторов  for (int i = 0; i < N; i++) {  h\_a[i] = 1.0f;  h\_b[i] = 2.0f;  }   // Выделение памяти на устройстве  cudaMalloc(&d\_a, size);  cudaMalloc(&d\_b, size);  cudaMalloc(&d\_c, size);   // Копирование данных на устройство  cudaMemcpy(d\_a, h\_a, size, cudaMemcpyHostToDevice);  cudaMemcpy(d\_b, h\_b, size, cudaMemcpyHostToDevice);   // Список размеров блоков для тестирования  int block\_sizes[] = {1, 16, 32, 64, 128, 256, 512, 1024};  int num\_sizes = sizeof(block\_sizes) / sizeof(int);   // События для замера времени  cudaEvent\_t start, stop;  cudaEventCreate(&start);  cudaEventCreate(&stop);   for (int i = 0; i < num\_sizes; ++i) {  int block\_size = block\_sizes[i];  int grid\_size = (N + block\_size - 1) / block\_size;   cudaEventRecord(start);  vectorAdd<<<grid\_size, block\_size>>>(d\_a, d\_b, d\_c, N);  cudaEventRecord(stop);  cudaEventSynchronize(stop);   float milliseconds = 0;  cudaEventElapsedTime(&milliseconds, start, stop);  printf("Block size: %4d, Time: %f ms\n", block\_size, milliseconds);   // Проверка ошибок  cudaError\_t err = cudaGetLastError();  if (err != cudaSuccess) {  printf("Error: %s\n", cudaGetErrorString(err));  }  }   // Копирование результата обратно  cudaMemcpy(h\_c, d\_c, size, cudaMemcpyDeviceToHost);   // Проверка корректности  bool correct = true;  for (int i = 0; i < N; i++) {  if (h\_c[i] != 3.0f) {  correct = false;  break;  }  }  printf("Result: %s\n", correct ? "Correct" : "Incorrect");   // Освобождение ресурсов  cudaFree(d\_a);  cudaFree(d\_b);  cudaFree(d\_c);  free(h\_a);  free(h\_b);  free(h\_c);  cudaEventDestroy(start);  cudaEventDestroy(stop);   return 0; } |
| --- |

Листинг 1 – программа для сложения векторов

1. Скомпилируем программу и запустим:

| ./vector\_add Block size: 1, Time: 14.740896 ms Block size: 16, Time: 0.235520 ms Block size: 32, Time: 0.137152 ms Block size: 64, Time: 0.105472 ms Block size: 128, Time: 0.091136 ms Block size: 256, Time: 0.079872 ms Block size: 512, Time: 0.095168 ms Block size: 1024, Time: 0.105472 ms Result: Correct |
| --- |

Листинг 2 - вывод программы

1. Перепишем программу, чтобы она использовала только одну конфигурацию выполнения нитей за раз:

#include <stdio.h>

#include <cuda\_runtime.h>

#define N (1 << 20)

#define SIZE 1

\_\_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() {

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

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

size\_t size = N \* sizeof(float);

// Выделение памяти на хосте

h\_a = (float\*)malloc(size);

h\_b = (float\*)malloc(size);

h\_c = (float\*)malloc(size);

// Инициализация векторов

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

h\_a[i] = 1.0f;

h\_b[i] = 2.0f;

}

// Выделение памяти на устройстве

cudaMalloc(&d\_a, size);

cudaMalloc(&d\_b, size);

cudaMalloc(&d\_c, size);

// Копирование данных на устройство

cudaMemcpy(d\_a, h\_a, size, cudaMemcpyHostToDevice);

cudaMemcpy(d\_b, h\_b, size, cudaMemcpyHostToDevice);

// События для замера времени

cudaEvent\_t start, stop;

cudaEventCreate(&start);

cudaEventCreate(&stop);

int block\_size = SIZE;

int grid\_size = (N + block\_size - 1) / block\_size;

cudaEventRecord(start);

vectorAdd<<<grid\_size, block\_size>>>(d\_a, d\_b, d\_c, N);

cudaEventRecord(stop);

cudaEventSynchronize(stop);

float milliseconds = 0;

cudaEventElapsedTime(&milliseconds, start, stop);

printf("Block size: %4d, Time: %f ms\n", block\_size, milliseconds);

// Проверка ошибок

cudaError\_t err = cudaGetLastError();

if (err != cudaSuccess) {

printf("Error: %s\n", cudaGetErrorString(err));

}

// Копирование результата обратно

cudaMemcpy(h\_c, d\_c, size, cudaMemcpyDeviceToHost);

// Проверка корректности

bool correct = true;

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

if (h\_c[i] != 3.0f) {

correct = false;

break;

}

}

printf("Result: %s\n", correct ? "Correct" : "Incorrect");

// Освобождение ресурсов

cudaFree(d\_a);

cudaFree(d\_b);

cudaFree(d\_c);

free(h\_a);

free(h\_b);

free(h\_c);

cudaEventDestroy(start);

cudaEventDestroy(stop);

return 0;

}

Листинг 3 - Программа, которая считает время выполнения складывания двух векторов с одной конфигурацией нитей за раз

1. Запустим nvprof для генерации отчета: