**ФГБОУ ВО   
Уфимский университет науки и технологий**

**Кафедра ВМиК**

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 100 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| 90 |  |  |  |  |  |  |  |  |  |  |  |  |
| 80 |  |  |  |  |  |  |  |  |  |  |  |  |
| 70 |  |  |  |  |  |  |  |  |  |  |  |  |
| 60 |  |  |  |  |  |  |  |  |  |  |  |  |
| 50 |  |  |  |  |  |  |  |  |  |  |  |  |
| 40 |  |  |  |  |  |  |  |  |  |  |  |  |
| 30 |  |  |  |  |  |  |  |  |  |  |  |  |
| 20 |  |  |  |  |  |  |  |  |  |  |  |  |
| 10 |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |

Параллельное сложение векторов на графическом процессоре

**ОТЧЕТ**

**по лабораторной работе**

**по** Технологиям параллельного программирования

(*наименование дисциплины*)

|  |
| --- |
| Лабораторная работа 4 |
| (обозначение документа) |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Группа |  |  | Фамилия, И., О. | Подпись | Дата | Оценка |
| МО-325Б |  |
|  |  |
| Студент | | | Шарыгин М.С. |  |  |  |
| Преподаватель | | | Спеле В.В. |  |  |  |
| Принял | | |  |  |  |  |

**Уфа 2025 г****.**

Содержание

[1 Цель работы 3](#_Toc193367759)

[2 Практическая часть 4](#_Toc193367760)

[2.1 Параметры графического процессора 4](#_Toc193367761)

[2.2 Таблицы 4](#_Toc193367762)

[2.3 Графики 4](#_Toc193367763)

[2.4 «NVIDIA Visual Profiler» 7](#_Toc193367764)

[3 Вывод 9](#_Toc193367765)

[4 Приложение 10](#_Toc193367766)

[4.1 Код программы с использованием «Unified Memory» 10](#_Toc193367767)

[4.2 Код программы без использования «Unified Memory» 11](#_Toc193367768)

# Цель работы

На примере задачи параллельного сложения векторов научиться разрабатывать простейшие параллельные программы средствами «CUDA C».

# Практическая часть

## Параметры графического процессора

Лабораторная работа выполнялась на компьютере в аудитории 1-407.

## Таблицы

Изображение выглядит как текст, снимок экрана, Шрифт, число

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 2.1 – Заполненные таблицы

## Графики

## «NVIDIA Visual Profiler»

Изображение выглядит как текст, программное обеспечение, веб-страница, снимок экрана

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 2.2 – «NVIDIA Visual Profiler» для программы с использованием «Unified Memory»

Изображение выглядит как снимок экрана, программное обеспечение, текст

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 2.3 – «NVIDIA Visual Profiler» для программы без использования «Unified Memory»

# Вывод

В ходе лабораторной работы мы на примере задачи параллельного сложения векторов научились разрабатывать простейшие параллельные программы средствами «CUDA C».

При увеличении количества нитей время расчета увеличивается, а производительность и пропускная способность падают, это происходит, скорее всего, из-за накладных расходов на управление потоками **–** каждый поток требует ресурсов для создания, переключения и синхронизации. Если количество потоков слишком велико, накладные расходы на управление ими могут превысить выгоду от параллельного выполнения задач. Это особенно заметно, если задачи небольшие, как, например, сложение двух векторов.

# Приложение

## Код программы с использованием «Unified Memory»

#define N 325000000

#define M 1024

#include <iostream>

#include "cuda\_runtime.h"

#include "device\_launch\_parameters.h"

\_\_global\_\_ void add(float\* a, float\* b, float\* c) {

int index = threadIdx.x + blockIdx.x \* blockDim.x;

if (index < N) c[index] = a[index] + b[index];

}

float random\_floats(float\* x, int count) {

for (int i = 0; i < count; i++)

x[i] = (rand() % 1000 - 500) / 1000.0;

return \*x;

}

void check\_results(float\* x, float\* y, float\* z, int count) {

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

if (abs(x[i] + y[i] - z[i]) > pow(10, -8))

return;

}

printf("Check completed correctly\n");

}

int main() {

float time = 0;

cudaEvent\_t start, stop;

cudaEventCreate(&start);

cudaEventCreate(&stop);

float\* a, \* b, \* c;

int size = N \* sizeof(float);

cudaMallocManaged((void\*\*)&a, size);

cudaMallocManaged((void\*\*)&b, size);

cudaMallocManaged((void\*\*)&c, size);

random\_floats(a, N);

random\_floats(b, N);

cudaEventRecord(start, 0);

add << <(N + M - 1) / M, M >> > (a, b, c);

cudaEventRecord(stop, 0);

cudaDeviceSynchronize();

cudaEventSynchronize(stop);

cudaEventElapsedTime(&time, start, stop);

check\_results(a, b, c, N);

printf("Elapsed time: %.2f ms\n", time);

cudaEventDestroy(start);

cudaEventDestroy(stop);

cudaFree(a);

cudaFree(b);

cudaFree(c);

return 0;

}

## Код программы без использования «Unified Memory»

#define N 325000000

#define M 1024

#include <iostream>

#include "cuda\_runtime.h"

#include "device\_launch\_parameters.h"

\_\_global\_\_ void add(float\* a, float\* b, float\* c) {

int index = threadIdx.x + blockIdx.x \* blockDim.x;

if (index < N) c[index] = a[index] + b[index];

}

void random\_floats(float\* x, int count) {

for (int i = 0; i < count; i++)

x[i] = (rand() % 1000 - 500) / 1000.0;

}

void check\_results(float\* x, float\* y, float\* z, int count) {

for (int i = 0; i < count; i++)

if (abs(x[i] + y[i] - z[i]) > pow(10, -8))

return;

printf("Check completed correctly\n");

}

int main() {

float time\_transaction1 = 0;

float time\_transaction2 = 0;

float time\_calculation = 0;

cudaEvent\_t start\_transaction1, stop\_transaction1, start\_transaction2, stop\_transaction2, start\_calculation, stop\_calculation;

cudaEventCreate(&start\_transaction1);

cudaEventCreate(&stop\_transaction1);

cudaEventCreate(&start\_transaction2);

cudaEventCreate(&stop\_transaction2);

cudaEventCreate(&start\_calculation);

cudaEventCreate(&stop\_calculation);

int size = N \* sizeof(float);

float\* gpu\_a;

float\* gpu\_b;

float\* gpu\_c;

float\* a = new float[size];

float\* b = new float[size];

float\* c = new float[size];

cudaMalloc((void\*\*)&gpu\_a, size);

cudaMalloc((void\*\*)&gpu\_b, size);

cudaMalloc((void\*\*)&gpu\_c, size);

random\_floats(a, N);

random\_floats(b, N);

cudaEventRecord(start\_transaction1, 0);

cudaMemcpy(gpu\_a, a, size, cudaMemcpyHostToDevice);

cudaMemcpy(gpu\_b, b, size, cudaMemcpyHostToDevice);

cudaEventRecord(stop\_transaction1, 0);

cudaEventRecord(start\_calculation, 0);

add << <(N + M - 1) / M, M >> > (gpu\_a, gpu\_b, gpu\_c);

cudaEventRecord(stop\_calculation, 0);

cudaEventRecord(start\_transaction2, 0);

cudaMemcpy(c, gpu\_c, size, cudaMemcpyDeviceToHost);

cudaEventRecord(stop\_transaction2, 0);

cudaDeviceSynchronize();

cudaEventElapsedTime(&time\_transaction1, start\_transaction1, stop\_transaction1);

cudaEventElapsedTime(&time\_transaction2, start\_transaction2, stop\_transaction2);

cudaEventElapsedTime(&time\_calculation, start\_calculation, stop\_calculation);

check\_results(a, b, c, N);

printf("Elapsed time\_transaction1: %.2f ms\n", time\_transaction1);

printf("Elapsed time\_transaction2: %.2f ms\n", time\_transaction2);

printf("Elapsed time\_calculation: %.2f ms\n", time\_calculation);

cudaEventDestroy(start\_transaction1);

cudaEventDestroy(stop\_transaction1);

cudaEventDestroy(start\_transaction2);

cudaEventDestroy(stop\_transaction2);

cudaEventDestroy(start\_calculation);

cudaEventDestroy(stop\_calculation);

cudaFree(gpu\_a);

cudaFree(gpu\_b);

cudaFree(gpu\_c);

delete[] a;

delete[] b;

delete[] c;

return 0;

}