Министерство науки и высшего образования Российской Федерации
Калужский филиал
федерального государственного автономного
образовательного учреждения высшего образования
«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»
(КФ МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ ***ИУК «Информатика и управление»***

КАФЕДРА ***ИУК4 «Программное обеспечение ЭВМ, информационные технологии»***

ДОМАШНЯЯ РАБОТА №1

«Программирование с использованием технологии NVIDIA CUDA»

ДИСЦИПЛИНА: «Параллельные процессы в информационных системах»

Выполнил: студент гр. ИУК4-31М _____ (Сафронов Н.С.)
(Подпись) (Ф.И.О.)

Проверил: _____ (Корнюшин Ю.П.)
(Подпись) (Ф.И.О.)

Дата сдачи (защиты):

Результаты сдачи (защиты):

- Балльная оценка:
- Оценка:

Калуга, 2025

Цель работы: сформировать практические навыки по использованию технологии NVIDIA CUDA, научиться компилировать и запускать программы, содержащие CUDA-код.

Задачи

Задачей в рамках выполнения домашней работы является написание программы согласно варианту, при этом необходимо реализовать 2 функции: одну для выполнения на процессоре, вторую для выполнения на видеокарте. Затем сравнить результаты (возвращаемые значения) и скорость работы.

Задание

Программа должна быть реализована с применением технологии NVIDIA CUDA.

Вариант 6

Нахождение суммы квадратов элементов строк матрицы для всех строк матрицы.

Результат выполнения работы

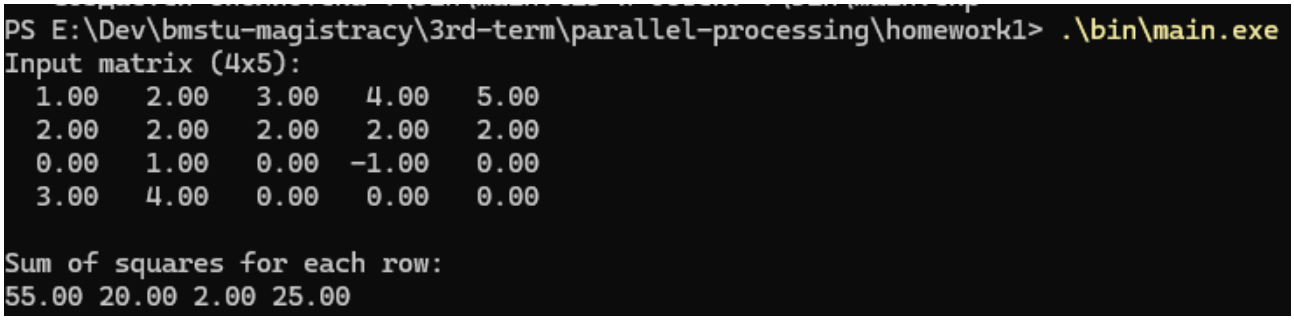

Рисунок 1 – Нахождение суммы квадратов элементов

Листинг программы

```
#include <cuda_runtime.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define CUDA_CHECK(call) \
    do { \
        cudaError_t err = call; \
        if (err != cudaSuccess) { \
            fprintf(stderr, "CUDA error at %s:%d - %s\n", __FILE__, __LINE__, \
                    cudaGetErrorString(err)); \
            exit(EXIT_FAILURE); \
        } \
    } while(0)

__global__ void sum_of_squares_kernel(const float* A, float* row_sums, int m,
int n) {
    int i = blockIdx.x * blockDim.x + threadIdx.x;
    if (i >= m) return;
```

```
    float sum = 0.0f;
    for (int j = 0; j < n; j++) {
        float val = A[i * n + j];
        sum += val * val;
    }
    row_sums[i] = sum;
}

void compute_row_sum_squares(const float* h_A, float* h_row_sums, int m, int n)
{
    size_t matrix_size = m * n * sizeof(float);
    size_t sums_size = m * sizeof(float);

    float *d_A = nullptr, *d_row_sums = nullptr;
    CUDA_CHECK(cudaMalloc(&d_A, matrix_size));
    CUDA_CHECK(cudaMalloc(&d_row_sums, sums_size));

    CUDA_CHECK(cudaMemcpy(d_A, h_A, matrix_size, cudaMemcpyHostToDevice));

    int threads_per_block = 256;
    int blocks = (m + threads_per_block - 1) / threads_per_block;
    sum_of_squares_kernel<<<blocks, threads_per_block>>>(d_A, d_row_sums, m, n);

    CUDA_CHECK(cudaGetLastError());
    CUDA_CHECK(cudaDeviceSynchronize());

    CUDA_CHECK(cudaMemcpy(h_row_sums, d_row_sums, sums_size,
cudaMemcpyDeviceToHost));

    CUDA_CHECK(cudaFree(d_A));
    CUDA_CHECK(cudaFree(d_row_sums));
}

void print_matrix(const float* A, int m, int n) {
    for (int i = 0; i < m; i++) {
        for (int j = 0; j < n; j++) {
            printf("%6.2f ", A[i * n + j]);
        }
        printf("\n");
    }
}

void print_vector(const float* v, int n) {
    for (int i = 0; i < n; i++) {
        printf("%.2f ", v[i]);
    }
    printf("\n");
}

int main() {
    const int m = 4;
    const int n = 5;

    float h_A[] = {
        1.0f, 2.0f, 3.0f, 4.0f, 5.0f,
        2.0f, 2.0f, 2.0f, 2.0f, 2.0f,
        0.0f, 1.0f, 0.0f, -1.0f, 0.0f,
        3.0f, 4.0f, 0.0f, 0.0f, 0.0f
    };

    float* h_row_sums = (float*)malloc(m * sizeof(float));

    printf("Input matrix (%dx%d):\n", m, n);
    print_matrix(h_A, m, n);
```

```
    compute_row_sum_squares(h_A, h_row_sums, m, n);

    printf("\nSum of squares for each row:\n");
    print_vector(h_row_sums, m);

    free(h_row_sums);
    return 0;
}
```

```
PS E:\Dev\bmstu-magistracy\3rd-term\parallel-processing\homework1> .\bin\gpu.exe
GPU Row Sum of Squares (CUDA)
Matrix size: 2000 x 3000
GPU computation time: 0.000456 seconds

First 5 row sums (GPU):
Row 0: 25335.91
Row 1: 24346.57
Row 2: 24717.97
Row 3: 25000.48
Row 4: 24863.01
```

Рисунок 2 – Выполнение программы на GPU

```
PS E:\Dev\bmstu-magistracy\3rd-term\parallel-processing\homework1> .\bin\cpu.exe
CPU Row Sum of Squares
Matrix size: 2000 x 3000
Computation time: 0.0145 seconds
First 5 row sums:
Row 0: 25335.91
Row 1: 24346.57
Row 2: 24717.97
Row 3: 25000.48
Row 4: 24863.01
```

Рисунок 3 – Выполнение программы на CPU

Листинг программы для CPU

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>

#ifdef _WIN32
    #include <windows.h>
    double get_time_sec() {
        LARGE_INTEGER freq, count;
        QueryPerformanceFrequency(&freq);
        QueryPerformanceCounter(&count);
        return (double)count.QuadPart / (double)freq.QuadPart;
    }
#else
    #include <sys/time.h>
    double get_time_sec() {
        struct timeval tv;
        gettimeofday(&tv, NULL);
        return tv.tv_sec + tv.tv_usec * 1e-6;
```

```
    }
#endif

void compute_row_sum_squares(const float* A, float* row_sums, int m, int n) {
    for (int i = 0; i < m; i++) {
        float sum = 0.0f;
        for (int j = 0; j < n; j++) {
            float val = A[i * n + j];
            sum += val * val;
        }
        row_sums[i] = sum;
    }
}

float* generate_random_matrix(int m, int n) {
    float* A = (float*)malloc(m * n * sizeof(float));
    if (!A) {
        fprintf(stderr, "Error: Failed to allocate matrix memory.\n");
        exit(EXIT_FAILURE);
    }

    srand(42);
    for (int i = 0; i < m * n; i++) {
        A[i] = ((float)rand() / RAND_MAX) * 10.0f - 5.0f;
    }
    return A;
}

int main(int argc, char* argv[]) {
    int m = 2000;
    int n = 3000;

    if (argc >= 3) {
        m = atoi(argv[1]);
        n = atoi(argv[2]);
        if (m <= 0 || n <= 0) {
            fprintf(stderr, "Error: Matrix dimensions must be positive
integers.\n");
            return EXIT_FAILURE;
        }
    }

    printf("CPU Row Sum of Squares\n");
    printf("Matrix size: %d x %d\n", m, n);

    float* A = generate_random_matrix(m, n);
    float* row_sums = (float*)calloc(m, sizeof(float));
    if (!row_sums) {
        fprintf(stderr, "Error: Failed to allocate result vector.\n");
        free(A);
        return EXIT_FAILURE;
    }

    double start = get_time_sec();
    compute_row_sum_squares(A, row_sums, m, n);
    double elapsed = get_time_sec() - start;

    printf("Computation time: %.4f seconds\n", elapsed);

    printf("First 5 row sums:\n");
    for (int i = 0; i < 5 && i < m; i++) {
        printf("Row %d: %.2f\n", i, row_sums[i]);
    }

    free(A);
    free(row_sums);
```

```
    return 0;
}
```

Листинг программы для GPU

```
#include <cuda_runtime.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>

void compute_row_sum_squares_cpu(const float* A, float* row_sums, int m, int n)
{
    for (int i = 0; i < m; i++) {
        float sum = 0.0f;
        for (int j = 0; j < n; j++) {
            float val = A[i * n + j];
            sum += val * val;
        }
        row_sums[i] = sum;
    }
}

__global__ void sum_of_squares_kernel(const float* A, float* row_sums, int m,
int n) {
    int i = blockIdx.x * blockDim.x + threadIdx.x;
    if (i >= m) return;

    float sum = 0.0f;
    for (int j = 0; j < n; j++) {
        float val = A[i * n + j];
        sum += val * val;
    }
    row_sums[i] = sum;
}

double compute_row_sum_squares_gpu(const float* h_A, float* h_row_sums, int m,
int n) {
    size_t matrix_size = m * n * sizeof(float);
    size_t sums_size = m * sizeof(float);

    float *d_A = nullptr, *d_row_sums = nullptr;
    cudaMalloc(&d_A, matrix_size);
    cudaMalloc(&d_row_sums, sums_size);

    cudaMemcpy(d_A, h_A, matrix_size, cudaMemcpyHostToDevice);

    int threads_per_block = 256;
    int blocks = (m + threads_per_block - 1) / threads_per_block;

    sum_of_squares_kernel<<<blocks, threads_per_block>>>(d_A, d_row_sums, m, n);
    cudaDeviceSynchronize();

    cudaEvent_t start, stop;
    cudaEventCreate(&start);
    cudaEventCreate(&stop);

    cudaEventRecord(start);
    sum_of_squares_kernel<<<blocks, threads_per_block>>>(d_A, d_row_sums, m, n);
    cudaEventRecord(stop);
    cudaEventSynchronize(stop);

    float milliseconds = 0;
    cudaEventElapsedTime(&milliseconds, start, stop);

    cudaMemcpy(h_row_sums, d_row_sums, sums_size, cudaMemcpyDeviceToHost);
```

```
    cudaFree(d_A);
    cudaFree(d_row_sums);
    cudaEventDestroy(start);
    cudaEventDestroy(stop);

    return milliseconds / 1000.0;
}

float* generate_random_matrix(int m, int n) {
    float* A = (float*)malloc(m * n * sizeof(float));
    srand(42);
    for (int i = 0; i < m * n; i++) {
        A[i] = ((float)rand() / RAND_MAX) * 10.0f - 5.0f;
    }
    return A;
}

int main() {
    const int m = 2000;
    const int n = 3000;

    printf("GPU Row Sum of Squares (CUDA)\n");
    printf("Matrix size: %d x %d\n", m, n);

    float* h_A = generate_random_matrix(m, n);
    float* h_sums_gpu = (float*)malloc(m * sizeof(float));
    float* h_sums_cpu = (float*)malloc(m * sizeof(float));

    double gpu_time = compute_row_sum_squares_gpu(h_A, h_sums_gpu, m, n);

    compute_row_sum_squares_cpu(h_A, h_sums_cpu, m, n);

    printf("GPU computation time: %.6f seconds\n", gpu_time);
    printf("\nFirst 5 row sums (GPU):\n");
    for (int i = 0; i < 5 && i < m; i++) {
        printf("Row %d: %.2f\n", i, h_sums_gpu[i]);
    }

    free(h_A);
    free(h_sums_gpu);
    free(h_sums_cpu);

    return 0;
}
```

Вывод: в ходе выполнения лабораторной работы были сформированы практические навыки по использованию технологии NVIDIA CUDA, компилированию и запуску программ, содержащих CUDA-код.