Министерство науки и высшего образования РФ
Федеральное государственное автономное
образовательное учреждение высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт космических и информационных технологий

Кафедра вычислительной техники

ОТЧЁТ О ПРАКТИЧЕСКОЙ РАБОТЕ №5

по дисциплине
«Гибридные вычислительные системы»

Преподаватель _____ С. А. Тарасов
подпись, дата инициалы, фамилия

Студент КИ22-07Б, 032212677 _____ Л. А. Глушков
номер группы, зачетной книжкой подпись, дата инициалы, фамилия

Студент КИ22-07Б, 032215583 _____ А. М. Коробков
номер группы, зачетной книжкой подпись, дата инициалы, фамилия

Студент КИ22-07Б, 032214653 _____ И. О. Бердин
номер группы, зачетной книжкой подпись, дата инициалы, фамилия

Красноярск 2025

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цель работы

Закрепить навык работы с разделяемой памятью и примитивами синхронизации нитей. Освоить программирование уровня warp’а и технику широковещания регистров.

Задание

1. Разработать кёрнел kernel_vecred_nobr (1), который принимает экземпляр VectorView по значению и вычисляет сумму элементов вектора. Для хранения частичных сумм блоков использовать разделяемую память.

2. Разработать кёрнел kernel_vecred_br (2), который принимает экземпляр VectorView по значению и вычисляет сумму элементов вектора, используя функцию __shfl_down_sync. Для хранения частичных сумм warp’ов использовать разделяемую память.

3. Используя фреймворк Google Test, разработать модульные тесты для функций (1) и (2) с размерами векторов $n \in \{1, 2, 3, 127, 129, 512, 541, 1037\}$. В качестве эталона для сравнения использовать результат аналогичной операции для Eigen::Matrix (sum); для верификации результатов применять макрос EXPECT_NEAR с абсолютной точностью 10^{-4}.

4. Используя фреймворк Google Benchmark, разработать бенчмарки для функций (1) и (2) с размерами векторов $n \in \{8 \cdot 2^0, 8 \cdot 2^1, 8 \cdot 2^2, ..., 8 \cdot 2^{28}\}$. Бенчмарки должны игнорировать время, затраченное на выделение, копирование и освобождение памяти. Для корректного измерения времени выполнения CUDA-кода необходимо использовать CUDA Events API.

5. Построить графики реальной вычислительной сложности для (1) и (2).

6. Построить график ускорения (2) относительно (1).

7. Объяснить экспериментальные результаты.

8. Подготовить отчёт, содержащий: • ключевые фрагменты кода; • ссылку на репозиторий с полной реализацией; • графики результатов измерений; • анализ и интерпретацию полученных результатов.

Ход работы

Ключевые фрагменты кода

Ключевые моменты кода:

- Реализация VectorOperations (исходный код приведен в приложении А);
- Реализация VectorOpearionsKernel (исходный код приведен в приложении А).

Результаты выполнения программы

На рисунках 1 и 2 представлены графики, основанные на результате выполнения программы.

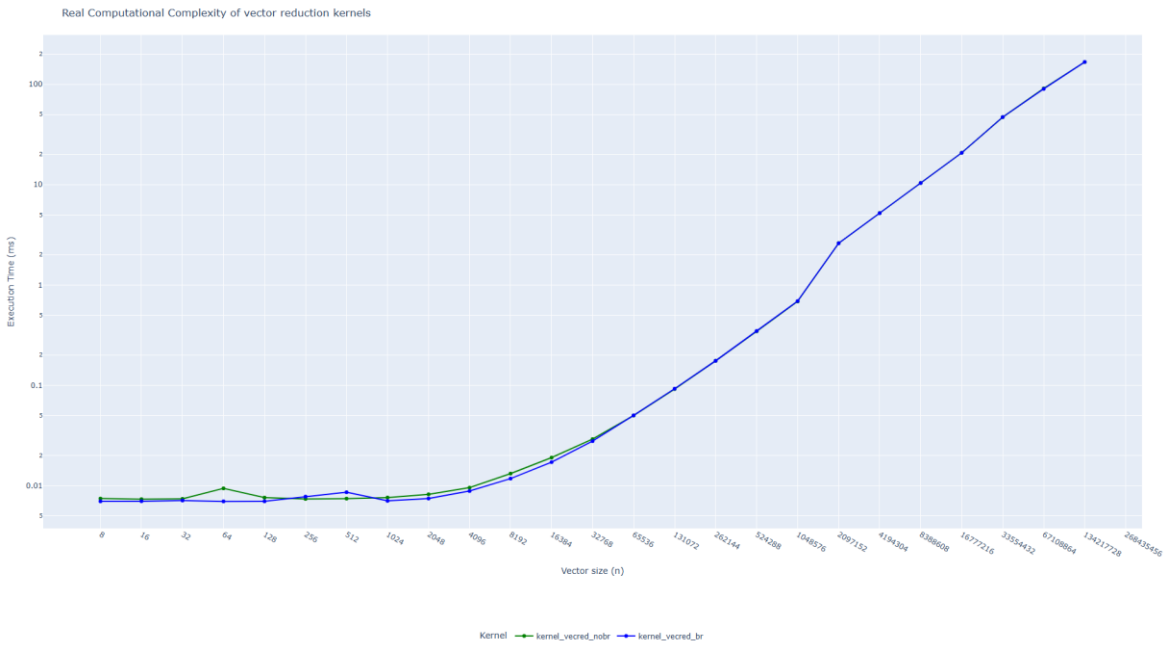

Рисунок 1 – График реальной вычислительной сложности

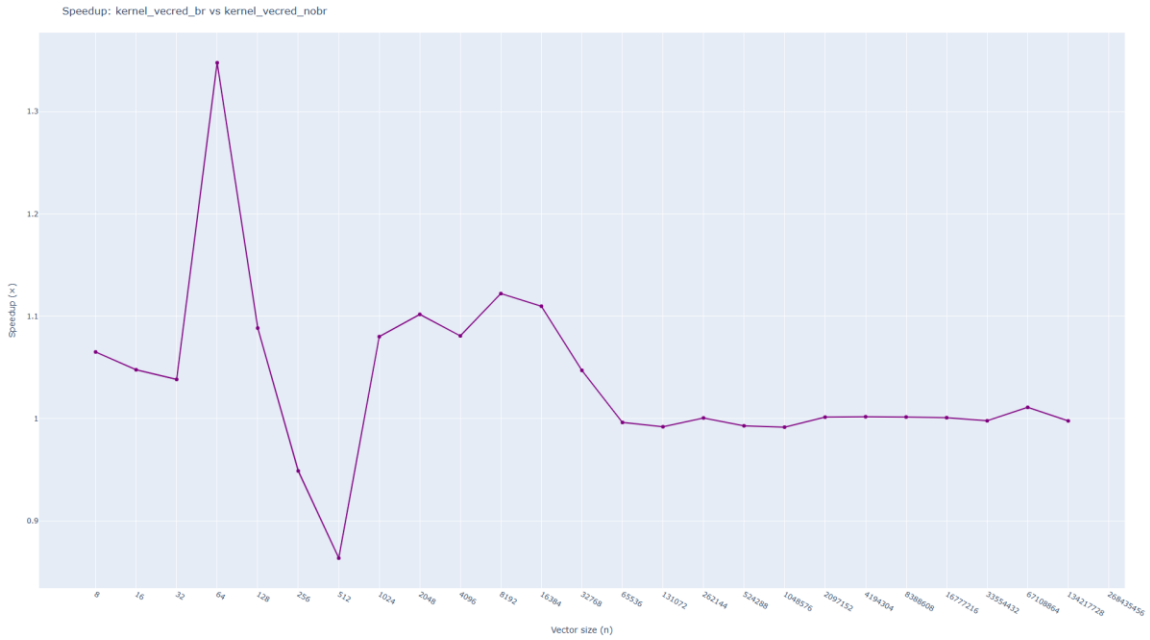

Рисунок 2 – График реальной ускорения

ЗАКЛЮЧЕНИЕ

В результате выполнения данной работы были закреплены навыки работы с разделяемой памятью и примитивами синхронизации нитей. Освоено программирование уровня warp’а и техника широковещания регистров.

ПРИЛОЖЕНИЕ А

```cpp
#pragma once
#include "vector.cuh"
#include "vectorOperationsKernel.cuh"

template<typename T, AddAlgorithm Algorithm>
T Vector<T, Algorithm>::sum() const {
    std::size_t n = view().size();

    if (n == 0) return static_cast<T>(0);

    constexpr std::size_t block_size = 256;

    std::size_t nBlocks = (((n + block_size - 1) / block_size) < 128) ? ((n + block_size - 1) / block_size) : 128;

    Data<T> blockSum(nBlocks);
    Data<T> result(1);

    T* blockSumData = blockSum.data();
    T* resultData = result.data();

    if constexpr (Algorithm == AddAlgorithm::nobr) {
        vectorAddNobrKernel<T><<<nBlocks, block_size, block_size * sizeof(T)>>> (view(), blockSumData, n);
    } else if constexpr (Algorithm == AddAlgorithm::br) {
        vectorAddBrKernel<T><<<nBlocks, block_size, ((block_size + 31) / 32) * sizeof(T)>>>(view(), blockSumData, n);
    }

    cudaError_t err = cudaGetLastError();
    if (err != cudaSuccess) {
        throw std::runtime_error("CUDA error 1: " + std::string(cudaGetErrorString(err)));
    }


    if constexpr (Algorithm == AddAlgorithm::nobr) {
        finalReductionNobrKernel<T><<<1, block_size, block_size * sizeof(T)>>>(blockSumData, resultData, n);
    } else if constexpr (Algorithm == AddAlgorithm::br) {
        finalReductionBrKernel<T><<<1, block_size, ((block_size + 31) / 32) * sizeof(T)>>>(blockSumData, resultData, n);
    }

    cudaError_t err2 = cudaGetLastError();
    if (err2 != cudaSuccess) {
        throw std::runtime_error("CUDA error 2: " + std::string(cudaGetErrorString(err2)));
    }

    T final;
    result.copyToHost(&final);

    cudaDeviceSynchronize();
    return final;
}
```

Рисунок А.1 – Реализация VectorOperations

```cpp
template<typename T>
__global__ void vectorAddNobrKernel(VectorView<T> a, T* blockSum, std::size_t n) {
    extern __shared__ T sharedMem[];

    std::size_t tid = threadIdx.x;
    std::size_t blockSize = blockDim.x;

    T threadSum = static_cast<T>(0);

    for (std::size_t i = tid; i < n; i += blockSize) {
        threadSum += a[i];
    }

    sharedMem[tid] = threadSum;
    __syncthreads();

    for (int i = blockSize / 2; i > 0; i >>= 1) {
        if (tid < i) {
            sharedMem[tid] += sharedMem[tid + i];
        }
        __syncthreads();
    }

    if (tid == 0) {
        blockSum[tid] = sharedMem[0];
    }
}
```

Рисунок А.2 – Реализация VectorOperationsKernel

```cpp
template<typename T>
__global__ void finalReductionNobrKernel(T* blockSum, T* sum, std::size_t n) {
        extern __shared__ T sharedMem[];
        std::size_t tid = threadIdx.x;

        if (tid < n) {
            sharedMem[tid] = blockSum[tid];
        } else {
            sharedMem[tid] = static_cast<T>(0);
        }
        __syncthreads();

        for (int i = blockDim.x / 2; i > 0; i >>= 1) {
            if (tid < i) {
                sharedMem[tid] += sharedMem[tid + i];
            }
            __syncthreads();
        }

        if (tid == 0) {
            *sum = sharedMem[0];
        }
}
```

Рисунок А.3 – Реализация VectorOperationsKernel

```cpp
template<typename T>
__global__ void vectorAddBrKernel(VectorView<T> a, T* blockSum, std::size_t n) {
    extern __shared__ T warpSum[];

    std::size_t tid = threadIdx.x;
    std::size_t blockSize = blockDim.x;
    std::size_t lane = tid % 32;
    std::size_t warp = tid / 32;
    std::size_t warpPerBlock = (blockDim.x + 31) / 32;

    T threadSum = static_cast<T>(0);
    for (std::size_t i = tid; i < n; i += blockSize) {
        threadSum += a[i];
    }

    T warpS = threadSum;
    unsigned mask = 0xFFFFFFFF;

    for (int j = 16; j > 0; j /=2) {
        warpS += __shfl_down_sync(mask, warpS, j);
    }

    if (lane == 0) {
        warpSum[warp] = warpS;
    }
    __syncthreads();

    if (warp == 0) {
        T val = (lane < warpPerBlock) ? warpSum[lane] : static_cast<T>(0);

        for (int i = 16; i > 0; i /= 2) {
            val += __shfl_down_sync(mask, val, i);
        }

        if (lane == 0) {
            blockSum[tid] = val;
        }
    }
}
```

Рисунок А.3 – Реализация VectorOperationsKernel

```cpp
template<typename T>
__global__ void finalReductionBrKernel(T* blockSum, T* sum, std::size_t n) {
    extern __shared__ T warpSum[];

    std::size_t tid = threadIdx.x;
    std::size_t blockSize = blockDim.x;
    std::size_t lane = tid % 32;
    std::size_t warp = tid / 32;
    std::size_t warpPerBlock = (blockDim.x + 31) / 32;

    T threadVal = static_cast<T>(0);
    if (tid < n) {
        threadVal = blockSum[tid];
    }

    T warpS = threadVal;
    unsigned mask = 0xFFFFFFFF;

    for (int j = 16; j > 0; j /= 2) {
        warpS += __shfl_down_sync(mask, warpS, j);
    }

    if (lane == 0) {
        warpSum[warp] = warpS;
    }
    __syncthreads();

    if (warp == 0) {
        T val = (lane < warpPerBlock) ? warpSum[lane] : static_cast<T>(0);

        for (int i = 16; i > 0; i /= 2) {
            val += __shfl_down_sync(mask, val, i);
        }

        if (lane == 0) {
            *sum = val;
        }
    }
}
```

Рисунок А.4 – Реализация VectorOperationsKernel