Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное
образовательное учреждение высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»
Высшая школы электроники и компьютерных наук
Кафедра системного программирования

ОТЧЕТ
о лабораторной работе №10
по дисциплине «Технологии параллельного программирования»

Выполнил:
студент группы КЭ-220
_____/Голенищев А. Б.
_____ 2024 г.

Отчет принял:
_____/Жулев А. Э.
_____ 2024 г.

Челябинск
2024

Настроили CUDA для сложения векторов, листнинг 1. Показан алгоритм из лекций, рисунок 1. Представлен основной код программы, листнинг 2. Проверили работоспособность программы, рисунок 2.

Рисунок 1. Алгоритм сложения векторов из лекции про технологии CUDA

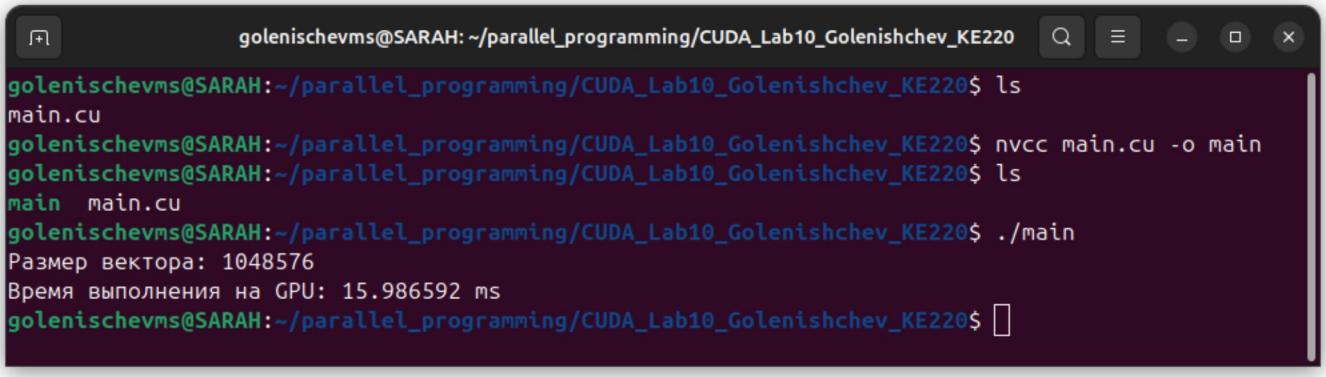

Рисунок 2. Результат выполнения программысложения векторов на ГП

```cpp
#include <stdio.h>
#include <cuda_runtime.h>

// CUDA kernel для сложения векторов
__global__ void vectorAddGPU(const float *a, const float *b, float *c, size_t N) {
    int idx = blockIdx.x * blockDim.x + threadIdx.x;
    if (idx < N) {
    c[idx] = a[idx] + b[idx];
    }
}
```

Листнинг 1. Код первой параллельной программы

Модифицируем программу, проведем дополнительные тесты, Рисунок 2.

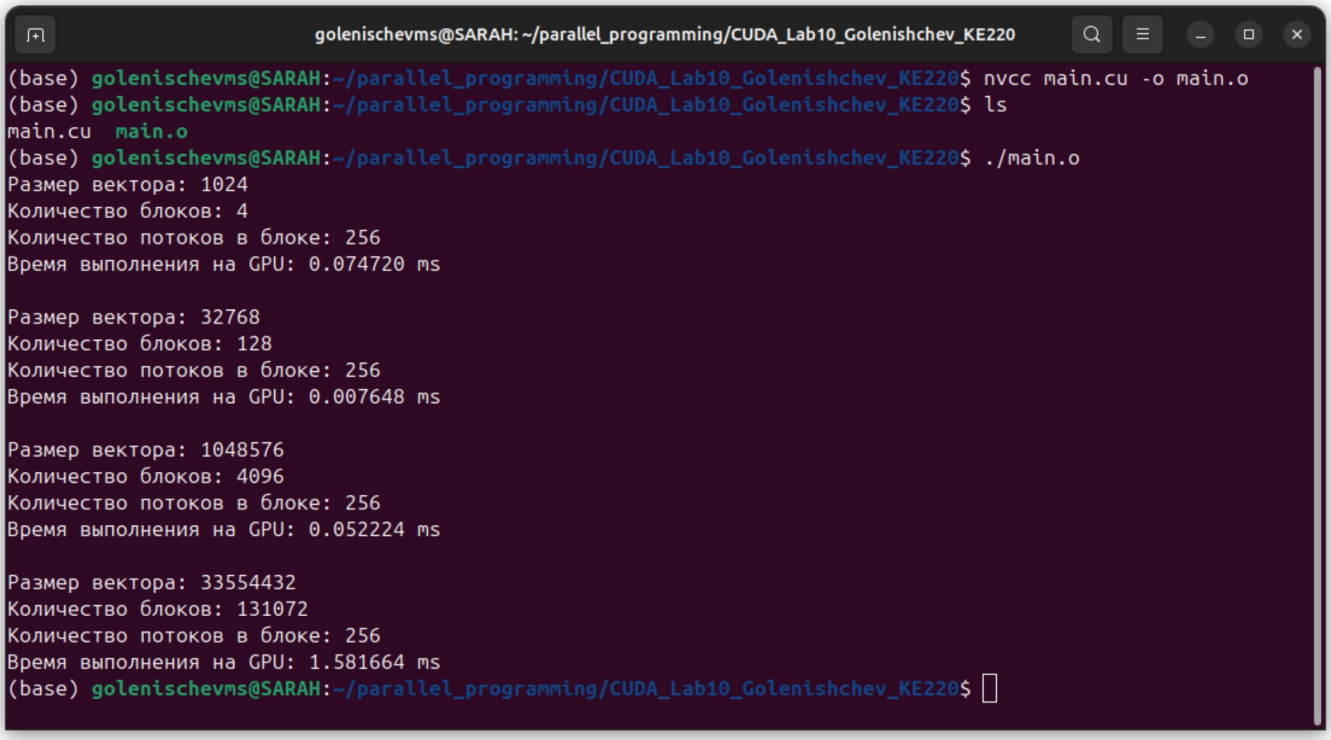

Рисунок 2. Результаты тестирования приложения

Программа выполняет сложение двух векторов на GPU с использованием CUDA. На хосте (CPU) создаются два входных вектора, которые инициализируются значениями, затем они копируются на устройство (GPU). Для сложения векторов используется CUDA-ядро, которое выполняется параллельно на множестве нитей, где каждая нить складывает соответствующие элементы из двух векторов и записывает результат в выходной вектор. Сетку и блоки нитей на GPU настраивают таким

образом, чтобы каждый элемент вектора обрабатывался одной нитью. После выполнения вычислений результаты копируются обратно на хост, и измеряется время работы на GPU с использованием CUDA событий.

Из приведённых данных можно сделать вывод, что время выполнения на GPU увеличивается с ростом размера вектора, однако рост времени не является линейным, что указывает на высокую эффективность параллельных вычислений. При небольших размерах вектора время выполнения включает накладные расходы на запуск ядра и передачу данных, что может занимать значительную долю, тогда как для больших векторов затраты на передачу данных и вычисления начинают доминировать, показывая прирост времени. Оптимальная производительность достигается при согласовании размеров блоков и потоков с архитектурой GPU.

Рисунок 3. График по результатам тестирования

```
int main() {
size_t N = 1 << 20; // Размер вектора (например, 1 миллион элементов)
size_t bytes = N * sizeof(float);
// Выделение памяти на хосте
float *h_a = (float *)malloc(bytes);
float *h_b = (float *)malloc(bytes);
float *h_c = (float *)malloc(bytes);
// Инициализация данных
for (size_t i = 0; i < N; ++i) {
h_a[i] = 1.0f; // Первый вектор
h_b[i] = 2.0f; // Второй вектор
}
// Выделение памяти на устройстве
float *d_a, *d_b, *d_c;
cudaMalloc(&d_a, bytes);
cudaMalloc(&d_b, bytes);
cudaMalloc(&d_c, bytes);
// Копирование данных с хоста на устройство
cudaMemcpy(d_a, h_a, bytes, cudaMemcpyHostToDevice);
cudaMemcpy(d_b, h_b, bytes, cudaMemcpyHostToDevice);
// Настройка сетки и блоков
int threads = 256; // Количество нитей в блоке
int blocks = (N + threads - 1) / threads; // Количество блоков
// Измерение времени выполнения на GPU
cudaEvent_t start, stop;
cudaEventCreate(&start);
cudaEventCreate(&stop);
cudaEventRecord(start);
vectorAddGPU<<<blocks, threads>>>(d_a, d_b, d_c, N);
cudaEventRecord(stop);
// Ожидание завершения всех вычислений на GPU
cudaDeviceSynchronize();
// Копирование результата обратно на хост
cudaMemcpy(h_c, d_c, bytes, cudaMemcpyDeviceToHost);
cudaEventSynchronize(stop);
float milliseconds = 0;
cudaEventElapsedTime(&milliseconds, start, stop);
printf("Размер вектора: %zu\n", N);
printf("Время выполнения на GPU: %f ms\n", milliseconds);
// Освобождение памяти
free(h_a);
free(h_b);
free(h_c);
cudaFree(d_a);
cudaFree(d_b);
cudaFree(d_c);
return 0;
}
```

Листнинг 2. Основной код программы

Выводы:

Изучили ключевые аспекты работы с параллельными вычислениями на GPU с использованием CUDA, включая основы организации вычислений в сетках и блоках нитей. Мы изучили, как эффективно переносить данные между CPU и GPU с помощью функций cudaMemcpy, а также как распределить задачи на множество потоков для ускорения вычислений. Программа показала, как можно использовать CUDA ядра для параллельного сложения двух векторов, что позволяет значительно ускорить вычисления по сравнению с последовательным подходом на CPU. Кроме того, была продемонстрирована настройка сетки блоков и нитей, а также использование CUDA-событий для точного измерения времени выполнения. В результате работы была получена практическая информация о том, как эффективно использовать возможности GPU для выполнения массивных параллельных вычислений.