Практическая работа №3

Тема: Реализация более сложных алгоритмов сортировки (слияние, быстрая, пирамидальная) на GPU с использованием CUDA

Цель работы

- Освоить программирование на CUDA и основы параллельных вычислений на GPU.
- Реализовать и оптимизировать сложные алгоритмы сортировки на GPU (слияние, быстрая сортировка, пирамидальная).
- Исследовать производительность алгоритмов на GPU и сравнить их с реализациями на CPU.

Теоретическая часть

Основные алгоритмы сортировки

1. Сортировка слиянием (Merge Sort):
   - Алгоритм "разделяй и властвуй", который разделяет массив на части, сортирует их и затем сливает.
   - Время выполнения: O(nlog n).

2. Быстрая сортировка (Quick Sort):
   - Алгоритм выбирает опорный элемент и разделяет массив на две части: элементы меньше опорного и элементы больше опорного, после чего рекурсивно сортирует каждую часть.
   - Среднее время выполнения: O(nlog n), но в худшем случае O(n^2).

3. Пирамидальная сортировка (Heap Sort):
   - Алгоритм строит бинарную кучу и последовательно извлекает наибольший элемент, перестраивая кучу.
   - Время выполнения: O(nlog n).

Программирование на CUDA

CUDA позволяет реализовывать параллельные вычисления, используя тысячи потоков. В этой практической работе мы реализуем три сортировки, используя следующие возможности **CUDA**:

- Индексация потоков: Каждый поток отвечает за обработку части массива.
- Использование разделяемой памяти: ускоряет обмен данными внутри одного блока, особенно важно для сортировки.

- Функции управления памятью: *cudaMalloc, cudaMemcpy и cudaFree* используются для выделения памяти на GPU, копирования данных и освобождения памяти.

Пример простой сортировки слиянием. Этот алгоритм использует рекурсивный подход "разделяй и властвуй" для сортировки:

```
//MergeSort
#include <iostream>
#include <vector>

// Функция для слияния двух отсортированных частей массива
void merge(std::vector<int>& arr, int left, int mid, int right) {
    int leftSize = mid - left + 1;  // Размер левой части
    int rightSize = right - mid;    // Размер правой части

    // Создаем временные массивы для левой и правой части
    std::vector<int> leftArr(leftSize);
    std::vector<int> rightArr(rightSize);

    // Копируем данные в временные массивы
    for (int i = 0; i < leftSize; ++i)
        leftArr[i] = arr[left + i];
    for (int j = 0; j < rightSize; ++j)
        rightArr[j] = arr[mid + 1 + j];

    // Инициализируем индексы для слияния
    int i = 0;         // Индекс для левой части
    int j = 0;         // Индекс для правой части
    int k = left;      // Индекс для слияния в основной массив

    // Сливаем левую и правую часть обратно в arr
    while (i < leftSize && j < rightSize) {
        if (leftArr[i] <= rightArr[j]) {
            arr[k] = leftArr[i];
            i++;
        } else {
            arr[k] = rightArr[j];
            j++;
        }
        k++;
    }

    // Копируем оставшиеся элементы левой части, если они есть
```

```
    while (i < leftSize) {
        arr[k] = leftArr[i];
        i++;
        k++;
    }

    // Копируем оставшиеся элементы правой части, если они есть
    while (j < rightSize) {
        arr[k] = rightArr[j];
        j++;
        k++;
    }
}

// Рекурсивная функция сортировки слиянием
void mergeSort(std::vector<int>& arr, int left, int right) {
    if (left < right) {
        int mid = left + (right - left) / 2;  // Находим середину
массива

        // Рекурсивно сортируем первую и вторую половину
        mergeSort(arr, left, mid);
        mergeSort(arr, mid + 1, right);

        // Сливаем отсортированные половины
        merge(arr, left, mid, right);
    }
}

int main() {
    std::vector<int> arr = {38, 27, 43, 3, 9, 82, 10};

    std::cout << "Исходный массив: ";
    for (int num : arr) {
        std::cout << num << " ";
    }
    std::cout << std::endl;

    // Запускаем сортировку слиянием
    mergeSort(arr, 0, arr.size() - 1);

    std::cout << "Отсортированный массив: ";
    for (int num : arr) {
```

```
        std::cout << num << " ";
    }
    std::cout << std::endl;

    return 0;
}
```

Практическая часть

Задание

1. Реализовать параллельную сортировку слиянием на CUDA:

- Разделите массив на блоки, каждый из которых будет обрабатываться одним блоком потоков.
- Сортируйте блоки параллельно и сливайте их по парам.

2. Реализовать параллельную быструю сортировку на CUDA:

- Используйте параллельные потоки для деления массива по опорному элементу.
- В каждом потоке выполняется быстрая сортировка на своей части массива.

3. Реализовать параллельную пирамидальную сортировку на CUDA:

- Постройте кучу и выполняйте извлечение элементов параллельно, где это возможно.

4. Сравнение производительности:

- Реализуйте последовательные версии этих алгоритмов на CPU.
- Измерьте время выполнения каждой сортировки на CPU и на GPU для массивов разного размера (например, 10,000, 100,000 и 1,000,000 элементов).
- Сравните производительность и сделайте выводы.

Контрольные вопросы

1. В чем различие между последовательной и параллельной реализациями сортировки слиянием?
2. Как распределение потоков и блоков влияет на производительность на CUDA?
3. Какие сложности возникают при реализации быстрой сортировки на GPU?
4. В каких случаях параллельная реализация сортировки на GPU может быть менее эффективной, чем на CPU?

5. Почему важно правильно выбирать размер блоков и потоков в CUDA?
6. Как использование разделяемой памяти может повлиять на производительность сортировки?
7. Что означает принцип "разделяй и властвуй" в контексте алгоритмов сортировки?