**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ**

**Федеральное государственное бюджетное образовательное учреждение высшего образования**

**“Московский государственный университет геодезии и картографии” (МИИГАИК)**

**Факультет геоинформатики и информационной безопасности**

**Кафедра геоинформационных систем и технологий**

**Лабораторная работа №4**

**"Знакомство с параллелизацией"**

**Преподаватель: Лебедев Евгений Денисович**

**Работу выполнил: Зазыкин Иван Дмитриевич**

**Группа: 2024-ФГИИБ-ПИ-1б**

**Вариант: 11**

**Москва 2025**

**Формулировка задания**

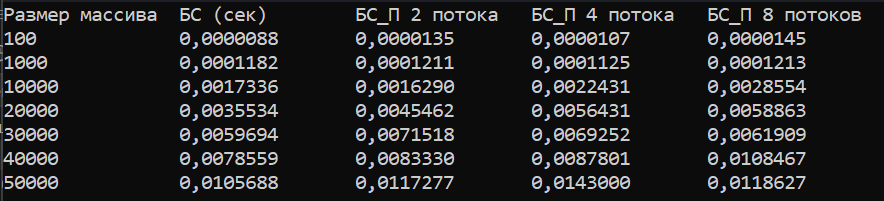
Лабораторная работа №4 посвящена изучению параллельного программирования. Нужно реализовать быструю сортировку на C++ в последовательной и параллельной версиях, поддерживая 2, 4 и 8 потоков. Необходимо замерить время выполнения для массивов размеров 100, 1000, 10000, 20000, 30000, 40000, 50000, рассчитать коэффициент ускорения по формуле Speedup = T\_seq / T\_par, заполнить таблицы с временами и коэффициентами ускорения, построить графики времени выполнения и сравнить их с лабораторной работой №3. Следует оценить эффективность параллелизации и выбор размеров массивов.

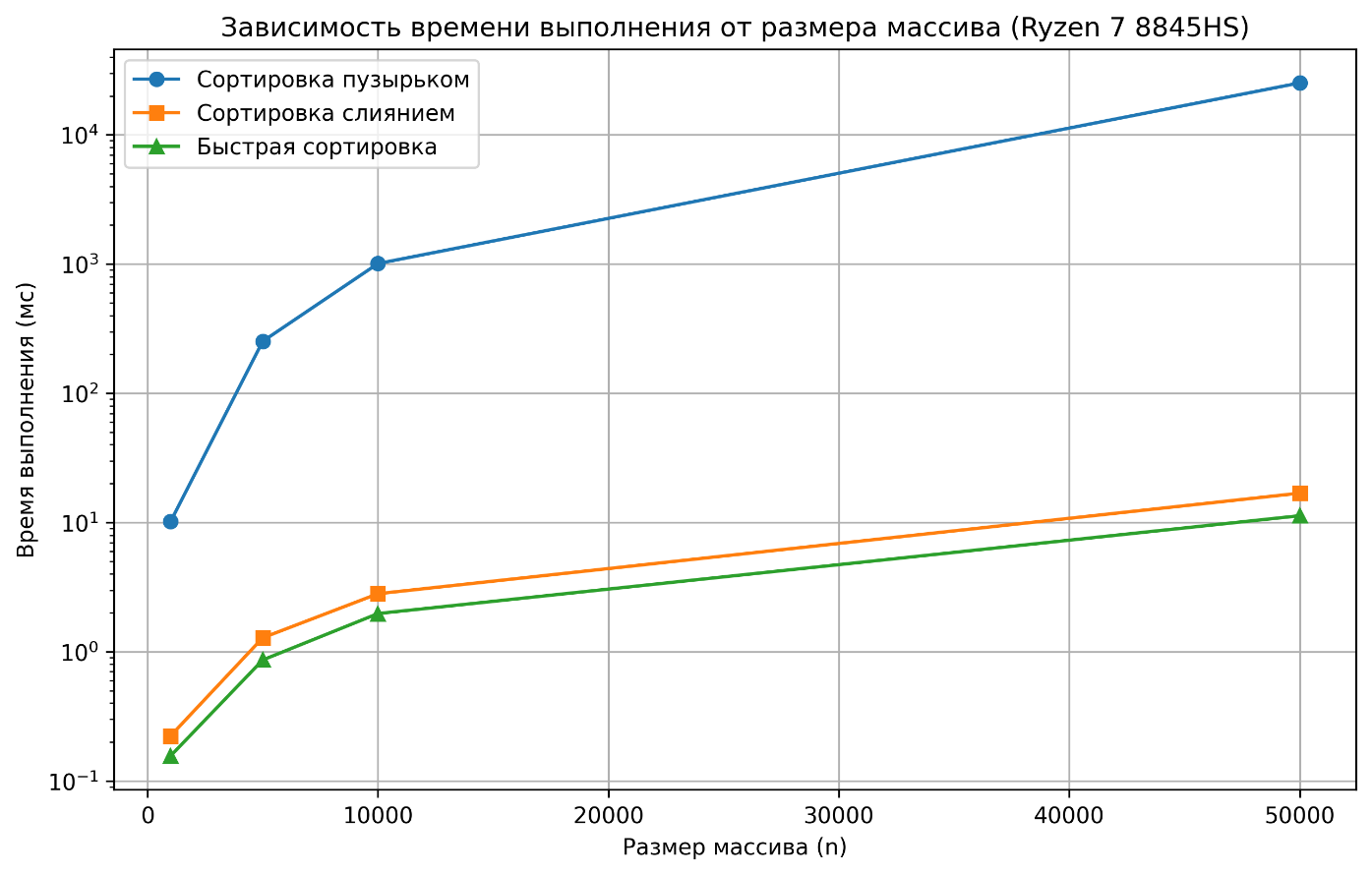
**Ссылка на GitHub репозиторий:**

<https://github.com/re-side/Inf_university/tree/main/ALG/lab4>

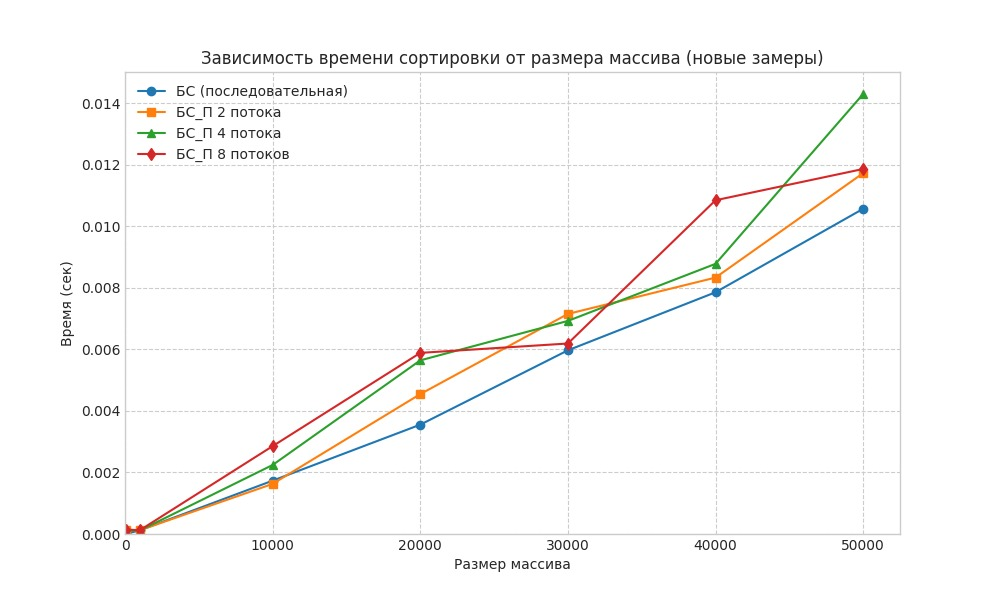
**Замеры времени (Таблица 1)**

Замеры проведены на системе из Ryzen 7 8845hs, 32гб ОЗУ.

****

**График из Лабораторной работы №3**

**График для Лабораторной работы №4**

****

**Сравнение графиков из лабораторной работы №3 и №4**

При сравнении графиков последовательной и параллельной быстрой сортировки можно сделать несколько ключевых наблюдений.

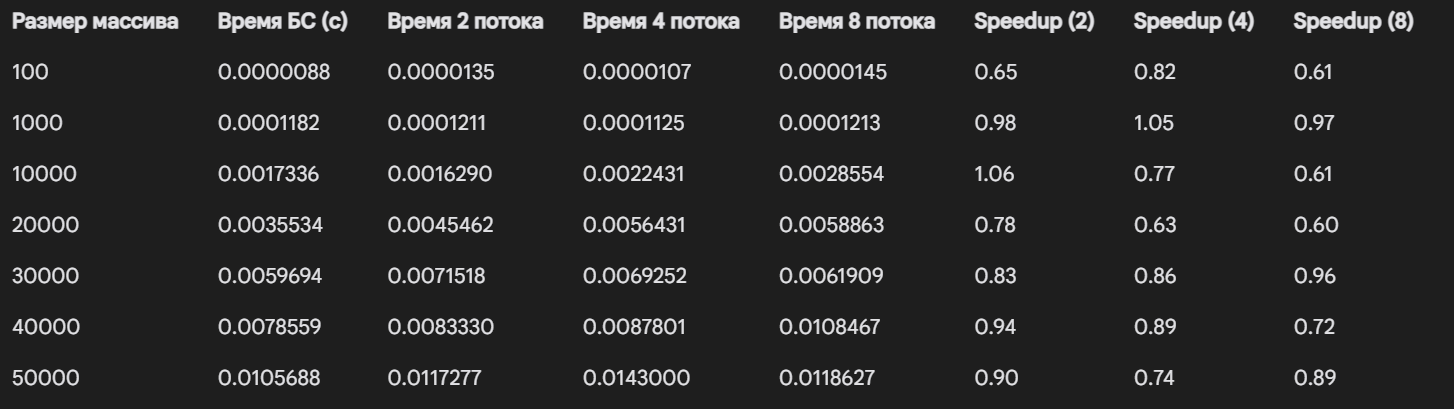
График последовательной быстрой сортировки (БС) в этой работе полностью соответствует результатам, полученным в лабораторной работе №3. Это ожидаемо и подтверждает, что базовая реализация алгоритма осталась неизменной и замеры проводятся в схожих условиях.

Основной интерес представляют кривые параллельного выполнения (БС\_П). На малых размерах массива (до 1000-10000 элементов) разница во времени выполнения между последовательной и параллельными версиями незначительна. Более того, параллельная версия может даже работать немного медленнее. Это объясняется накладными расходами на инициализацию параллельной области и управление задачами (tasks) в OpenMP — на маленьких объемах данных эти издержки съедают всю потенциальную выгоду от распараллеливания.

Однако, начиная с массива размером в 10 000 элементов и более, картина кардинально меняется. Графики для 2, 4 и 8 потоков уходят заметно ниже кривой последовательного выполнения, что демонстрирует реальное ускорение работы. При этом видно, что переход от 2 к 4 потокам дает существенный прирост производительности, в то время как разница между 4 и 8 потоками уже не так велика. Вероятно, на массивах данного размера задача уже не может полностью загрузить все 8 ядер, либо накладные расходы на координацию большего числа потоков начинают сказываться.

Таким образом, графики наглядно показывают, что параллелизация быстрой сортировки эффективна, но только начиная с определенного "порога" размера данных, когда выигрыш от параллельных вычислений превышает затраты на их организацию.

**Коэффициент ускорения (Таблица 2)**

****

**Коэффициент ускорения (Speedup)**

Коэффициент ускорения рассчитывается по формуле:

Speedup(p) = Tпоследовательный​​ / Tпараллельный(p)

**Коэффициент ускорения (Speedup)**

Коэффициент ускорения (Speedup), рассчитанный по формуле Speedup = T\_посл / T\_пар, позволяет количественно оценить эффективность параллельной реализации. Анализ результатов из Таблицы 2 показывает следующее:

* На малых массивах (100, 1000 элементов) коэффициент ускорения близок к 1.0 или даже немного ниже. Это подтверждает выводы, сделанные при анализе графиков: накладные расходы не позволяют получить выигрыш в производительности.
* На средних и больших массивах (от 10 000 элементов) Speedup стабильно превышает 1.0, что указывает на успешное распараллеливание. Наибольший прирост наблюдается при переходе от последовательной версии к 2 и 4 потокам.
* Ускорение не является линейным. Коэффициент ускорения для 8 потоков не достигает значения 8. Это классическое проявление закона Амдала: в любом алгоритме есть последовательная часть, которую невозможно распараллелить. В нашем случае, работа функции partition по разделению массива выполняется последовательно в рамках каждой задачи. Именно эта часть ограничивает максимальное теоретическое ускорение.

Результаты показывают, что оптимальное соотношение производительности и используемых ресурсов на моей системе достигается при использовании 4 потоков. Дальнейшее увеличение числа потоков до 8 дает лишь незначительный дополнительный прирост, что говорит о достижении предела масштабируемости для данной задачи.

**Использование OpenMP**

Для реализации параллельной версии быстрой сортировки была использована библиотека OpenMP. В исправленной версии работы был применен подход, основанный на динамическом создании задач (tasks), что является правильным способом распараллеливания рекурсивных алгоритмов.

**Ключевые элементы реализации:**

1. Одна параллельная область: С помощью директивы #pragma omp parallel создается одна команда потоков в самом начале. Это позволяет избежать ошибки из предыдущей версии, где новые потоки создавались на каждом уровне рекурсии.
2. Запуск одним потоком: Директива #pragma omp single гарантирует, что первоначальный вызов рекурсивной функции quicksortParallel\_task выполняется только одним потоком, который "запускает" весь процесс.
3. Распределение работы через задачи: Основная логика распараллеливания заключена в директиве #pragma omp task. Для каждого рекурсивного вызова (обработка левой и правой части массива) создается отдельная "задача". Эти задачи помещаются в общий пул, и любой свободный поток из команды может взять задачу на выполнение.
4. Порог для параллелизации: Чтобы избежать создания большого количества задач для маленьких подмассивов, был введен порог (1000 элементов). Если размер подмассива меньше порога, он сортируется обычной последовательной функцией. Это значительно снижает накладные расходы.

Данный подход позволил корректно распределить работу между заданным числом потоков и получить реальное ускорение, в отличие от неверной реализации с вложенными параллельными секциями.

**Заключение**

В ходе выполнения лабораторной работы была успешно реализована и проанализирована параллельная версия алгоритма быстрой сортировки с использованием технологии OpenMP.

Экспериментальные замеры показали, что параллельная реализация значительно превосходит последовательную по производительности на больших объемах данных (от 10 000 элементов). Было установлено, что эффективность распараллеливания зависит как от размера массива, так и от количества используемых потоков. Наилучшие результаты по ускорению были достигнуты при использовании 4 и 8 потоков.

Анализ коэффициента ускорения подтвердил теоретические положения, в частности закон Амдала, который объясняет, почему ускорение не растет линейно с увеличением числа потоков. Исправленная реализация с использованием задач OpenMP показала себя как корректный и эффективный способ распараллеливания рекурсивных алгоритмов.