Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н. Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н. Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

ОТЧЕТ

по лабораторной работе №4

по курсу «Анализ Алгоритмов»

на тему: «Параллельные вычисления на основе нативных потоков»

Студент ИУ7-51Б (Группа) _____ (Подпись, дата) Савинова М. Г. (Фамилия И. О.)

Преподаватель _____ (Подпись, дата) Волкова Л. Л. (Фамилия И. О.)

2024 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Многопоточность представляет собой способность процессора выполнять несколько задач или потоков одновременно, что обеспечивается операционной системой. Этот подход отличается от многопроцессорности, где каждый процессор выполняет отдельную задачу.

При последовательной реализации алгоритма только одно ядро процессора используется для выполнения программы. Однако при использовании параллельных вычислений — многопоточности — разные ядра могут одновременно решать независимые вычислительные задачи, что приводит к ускорению общего решения задачи [1].

Целью данной лабораторной работы является получение навыков организации параллельного выполнения операций.

Для достижения поставленной цели необходимо выполнить следующие задачи:

1) описать алгоритм сортировки слиянием;
2) разработать версии приведенного алгоритма при использовании одного и нескольких потоков;
3) определить средства программной реализации;
4) реализовать разработанные алгоритмы;
5) выполнить замеры процессорного времени работы различных реализаций алгоритма;
6) провести сравнительный анализ зависимостей времени решения задачи от размерности входа и количества вспомогательных потоков.

1 Аналитическая часть

В данной части работы будет описан алгоритм сортировки слиянием, а также рассмотрено использование многопоточности при его реализации.

1.1 Описание алгоритма

Алгоритм сортировки слиянием применяет принцип «разделяй и властвуй» для упорядочивания элементов в массиве.

Алгоритм сортировки состоит из нескольких этапов.

1) *Разделение*: исходный массив разделяется на две равные (или почти равные) половины. Это делается путем нахождения середины массива и создания двух новых массивов, в которые будут скопированы элементы из левой и правой половин.

2) *Рекурсивная сортировк*а: каждая половина массива рекурсивно сортируется с помощью алгоритма сортировки слиянием. Этот шаг повторяется до тех пор, пока размер каждой половины не станет равным 1.

3) *Слияние*: отсортированные половины массива объединяются в один отсортированный массив. Для этого создается новый массив, в который будут последовательно добавляться элементы из левой и правой половин. При добавлении элементов выбирается наименьший элемент из двух половин и добавляется в новый массив. Этот шаг повторяется до тех пор, пока все элементы не будут добавлены в новый массив [2].

1.2 Сложность алгоритма

Обозначим размер сортируемого массива как n. Алгоритм сортировки слиянием разделяет массив на две половины до тех пор, пока размер каждой не станет равным 1. Это занимает $O(\log(n))$ итераций. На каждом шаге происходит слияние двух отсортированных половин, что занимает $O(n)$ времени. Так как слияние происходит на каждом шаге, общее время слияния будет $O(n \cdot \log(n))$. Таким образом, общее время выполнения алгоритма сортировки слиянием занимает $O(n \cdot \log(n))$.

Алгоритм сортировки слиянием требует дополнительной памяти для хранения временных массивов при разделении и слиянии. Размер временных

массивов равен размеру исходного массива [2].

1.3 Использование потоков

В данной задаче возможно использование потоков на 2 этапе алгоритма. Создание потока возможно на каждый шаг рекурсии: в таком случае отдельный поток будет рекурсивно сортировать отдельную часть массива и затем выполнять ее слияние. Поток, запустивший следующий шаг рекурсии, выделяет на каждую половину массива отдельный поток и ждет окончания работы потоков, сортирующих 2 части массива.

Так как данные массива изменяются только на шаге слияния и сортировки отдельных частей, а рекурсивная сортировка разбивает массив на непересекающиеся части, в использовании средств синхронизации в виде мьютекса нет необходимости.

Вывод

В данной части работы был описан алгоритм сортировки слиянием и рассмотрено использование многопоточности в его реализации.

2 Конструкторская часть

В данной части работы будут рассмотрены схемы алгоритмов различных реализаций сортировки слиянием.

2.1 Требования к программному обеспечению

К программе предъявлен ряд требований:

— наличие интерфейса для выбора действий;

— динамическое выделение памяти под массив данных;

— работа с массивами и «нативными» потоками.

2.2 Требования к временным замерам

Процессорное время — это время, которое потратил процессор на выполнение задачи. Реальное время — время, прошедшее с начала выполнения задачи [3].

В данной работе, при использовании многопоточности возможно ожидание одними потоками выполнения других потоков. В это время поток не выполняет никаких действий, поэтому простой не влияет на процессорное время, однако это влияет на результирующее реальное время. Таким образом для корректного сравнения различных реализаций по времени работы стоит замерять и сравнивать реальное время выполнения реализаций алгоритмов.

2.3 Схемы алгоритмов

На рисунках 2.1 – 2.3 приведены схемы алгоритмов различных вариаций алгоритма сортировки слиянием

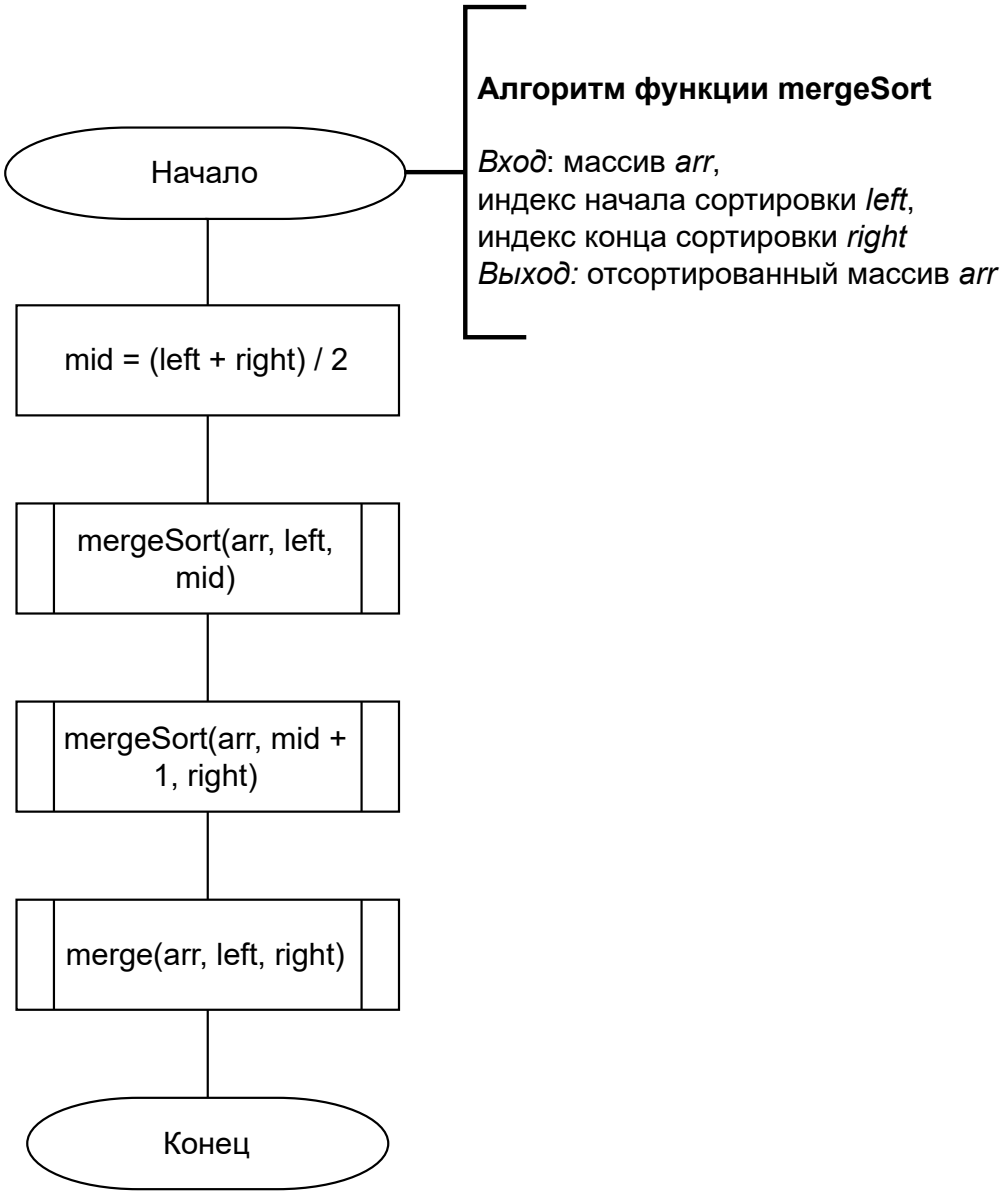

Рисунок 2.1 – Схема алгоритма сортировки слиянием при использовании одного потока

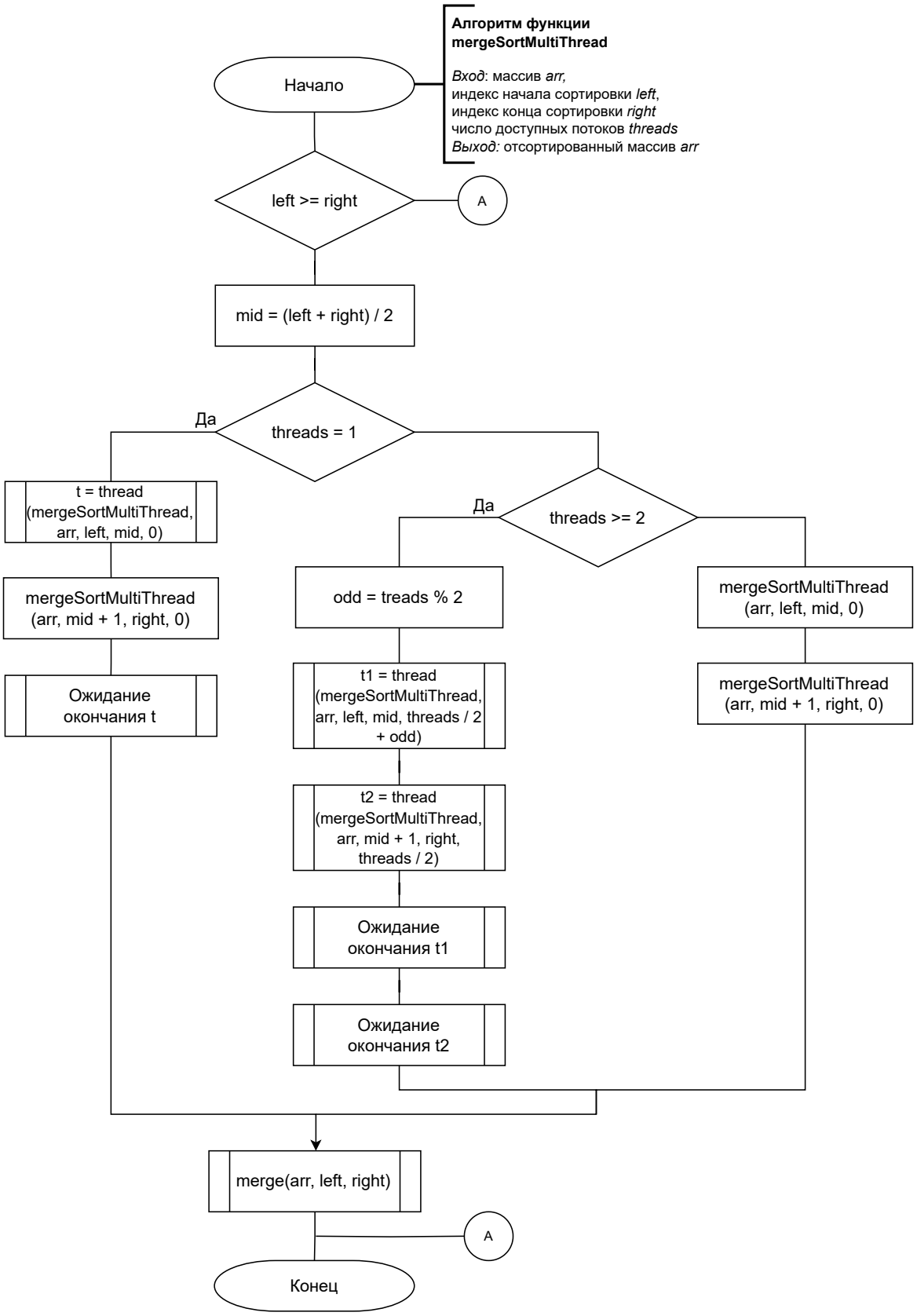

Рисунок 2.2 – Схема алгоритма сортировки слиянием при использовании нескольких потоков

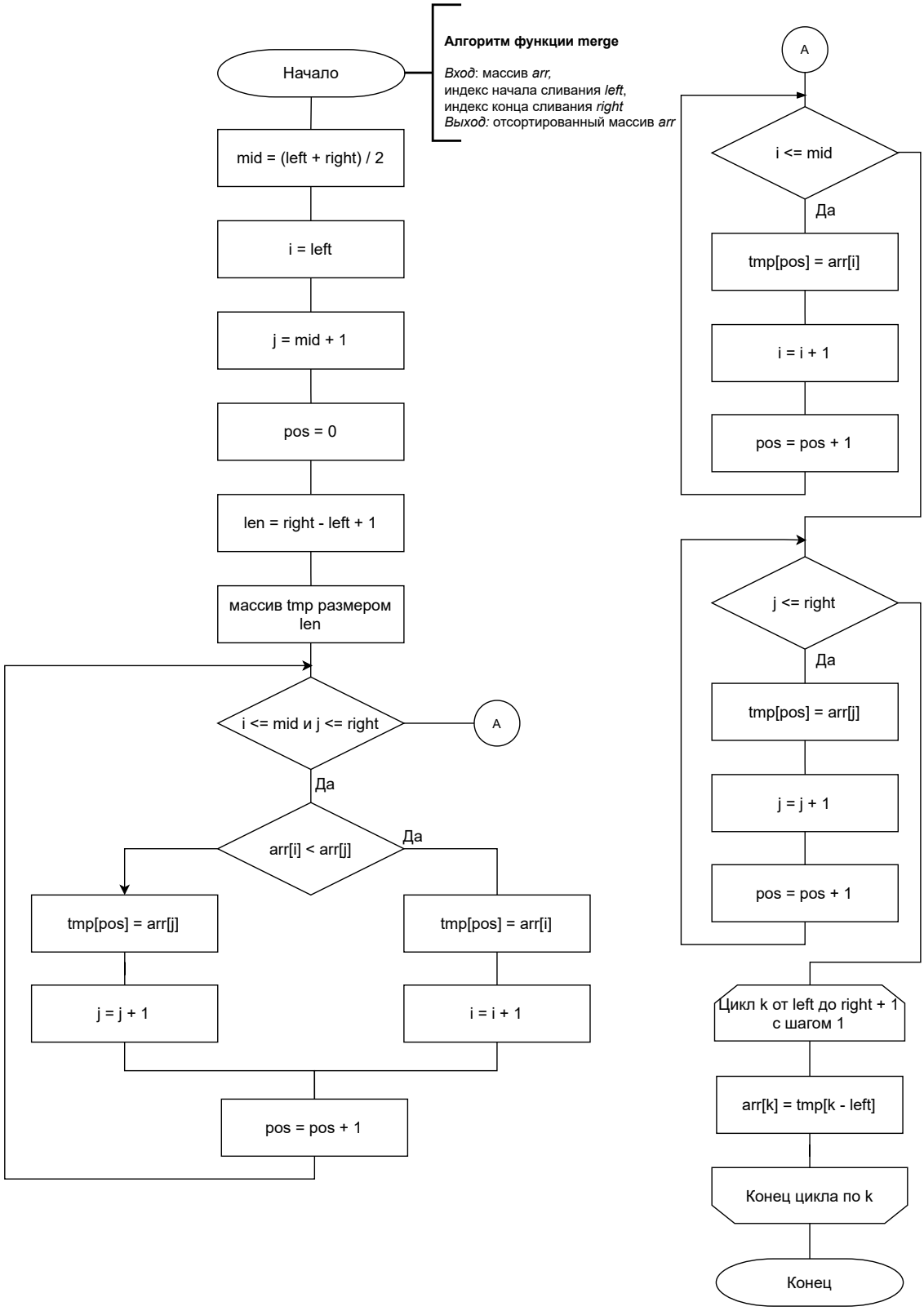

Рисунок 2.3 – Схема алгоритма слияния отсортированных последовательностей

Алгоритм слияния отсортированных последовательностей используется во всех рассматриваемых версиях алгоритма; в случае использования нескольких потоков, слияние будет происходить в отдельном потоке.

При использовании нескольких потоков (схема на рис. 2.2), число доступных потоков делится на 2 при каждом шаге рекурсии — в таком случае число доступных потоков поровну разделяется при каждом разбиении массива на подмассивы для слияния.

Вывод

В данном разделе были построены схемы рассматриваемых алгоритмов.

3 Технологический раздел

В данной части работы будут описаны средства реализации программы, а также листинги и модульные тесты.

3.1 Средства реализации

Алгоритмы для данной лабораторной работы были реализованы на языке C++, так как в библиотеке приведенного языка присутствует функция clock, которая позволяет получить количество тиков с времени начала выполнения программы, при делении полученного значения на макропеременную CLOCKS_PER_SEC возможно получение значения времени в секундах [4].

Для создания потоков и работы с ними был использован класс thread из стандартной библиотеки выбранного языка.

В листинге 3.1, приведен работы с описанным классом, каждый объект класса представляет собой поток операционной системы, что позволяет нескольким функциям выполняться параллельно [5].

Листинг 3.1 – Пример работы с классом thread

```
#include <iostream>
#include <thread>

void foo(int a)
{
    std::cout << a << '\n';
}

int main()
{
    std::thread thread(foo, 10);
    thread.join();

    return 0;
}
```

Поток начинает свою работу после создания объекта, соответствующего класса, запуская функцию, приведенную в его конструкторе [5]. В данном примере будет запущен 1 поток, который выполнит функцию foo, которая выведет число 10 на экран.

3.2 Сведения о модулях программы

Данная программа разбита на следующие модули:

— `main.cpp` — файл, содержащий точку входа в программу;

— `algs.cpp` — файл, содержащий реализацию алгоритма сортировки слиянием;

— `measure.cpp` — файл, содержащий функции, замеряющие процессорное время выполнения алгоритмов сортировок.

3.3 Реализация алгоритмов

Листинги А.1 – А.3 исходных кодов алгоритмов приведены в приложении А.

3.4 Тестирование

В таблице 3.1 приведены модульные тесты для разработанных алгоритмов сортировки. Все приведенные массивы были отсортированы с помощью сортировки слиянием, в столбце «Один поток» показаны результаты использования реализации с 1 потоком, в столбце «Несколько потоков» показаны результаты использования реализации с несколькими потоками. Все тесты пройдены успешно.

Таблица 3.1 – Модульные тесты

Массив	**Размер**	**Ожидаемый р-т**	**Фактический результат**	
			Один поток	**Несколько потоков**
1 2 3 4	4	1 2 3 4	1 2 3 4	1 2 3 4
4 3 2 1	4	4 3 2 1	4 3 2 1	4 3 2 1
3 5 1 6	4	1 3 5 6	1 3 5 6	1 3 5 6
-5 -1 -3 -4 -2	5	-5 -4 -3 -2 -1	-5 -4 -3 -2 -1	-5 -4 -3 -2 -1
1 -3 2 9 -9	5	-9 -3 1 2 9	-9 -3 1 2 9	-9 -3 1 2 9

Вывод

В данной части работы были представлены листинги реализованных алгоритмов и тесты, успешно пройденные программой.

4 Исследовательский раздел

В данном разделе будут приведены примеры работы программ, постановка исследования и сравнительный анализ алгоритмов на основе полученных данных.

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялись замеры по времени, представлены далее.

— Процессор: AMD Ryzen 5 5500U – 2.10 ГГц;

— Оперативная память: 16 ГБайт;

— Операционная система: Windows 10 Pro 64-разрядная система версии 22H2.

При замерах времени ноутбук был включен в сеть электропитания и был нагружен только системными приложениями.

4.2 Демонстрация работы программы

На рисунке 4.1 представлена демонстрация работы разработанного ПО.

```
                Меню
1. Обычная сортировка слиянием
2. Расспаралл. сортировка слиянием

3. Замерить время
0. Выход

Выберете пункт (0-4): 1

Длина массива: 5
5
4
3
2
1
До: 5 4 3 2 1
После: 1 2 3 4 5

                Меню
1. Обычная сортировка слиянием
2. Расспаралл. сортировка слиянием

3. Замерить время
0. Выход

Выберете пункт (0-4): 2

Длина массива: 5
9
8
7
6
5
До: 9 8 7 6 5
Количество потоков: 2
После: 5 6 7 8 9
```

Рисунок 4.1 – Демонстрация работы программы

4.3 Временные характеристики

Результаты замеров времени сортировки массива для разного числа потоков вычисления приведены в таблице 4.1.

Таблица 4.1 – Полученная таблица замеров по времени различных реализаций алгоритмов сортировки

n	Один поток(мс)	Несколько потоков(мс)	Число потоков
50000	58.886	58.933	0
50000	58.886	46.289	2
50000	58.886	52.604	4
50000	58.886	54.353	6
50000	58.886	58.822	8
50000	58.886	57.853	10
50000	58.886	61.572	12
50000	58.886	64.308	14
50000	58.886	64.89	16
50000	58.886	67.216	18
50000	58.886	67.029	20
50000	58.886	68.892	22
50000	58.886	70.15	24
50000	58.886	72.656	26
50000	58.886	73.546	28
50000	58.886	72.768	30
50000	58.886	73.515	32
50000	58.886	75.199	34
50000	58.886	75.746	36
50000	58.886	76.953	38
50000	58.886	79.806	40
50000	58.886	77.823	42
50000	58.886	79.456	44
50000	58.886	81.63	46
50000	58.886	81.089	48

Для таблицы 4.1 расчеты проводились с шагом изменения числа дополнительных потоков 2, сортировки производились 100 раз, после чего результат усредняется. В качестве сортируемых значений использовались числа от 1 до 10000000. Данные генерировались из равномерного распределения.

Значение столбца «n» определяет число сортируемых элементов в массиве; значение из столбца «Один поток» показывает результат замеров работы реализации при использовании 1 потока; значение из столбца «Несколько потоков» показывают результаты замеров работы многопоточной реализации. «Число потоков» определяет число вспомогательных потоков, для получения результата при многопоточной реализации алгоритма. Результаты замеров

времени приведен в миллисекундах.

По таблице 4.1 был построен график, (см. рисунок 4.2). Также была получена таблица 4.2, в которой в многопоточной реализации был использован 1 вспомогательный поток: данное число было выбрано для демонстрации уменьшения времени получения результата при использовании многопоточности с минимальным числом вспомогательных потоков.

По таблице 4.2 был получен график, представленный на рисунке 4.3. Обозначения столбцов и условия получения данных аналогичны таблице 4.1.

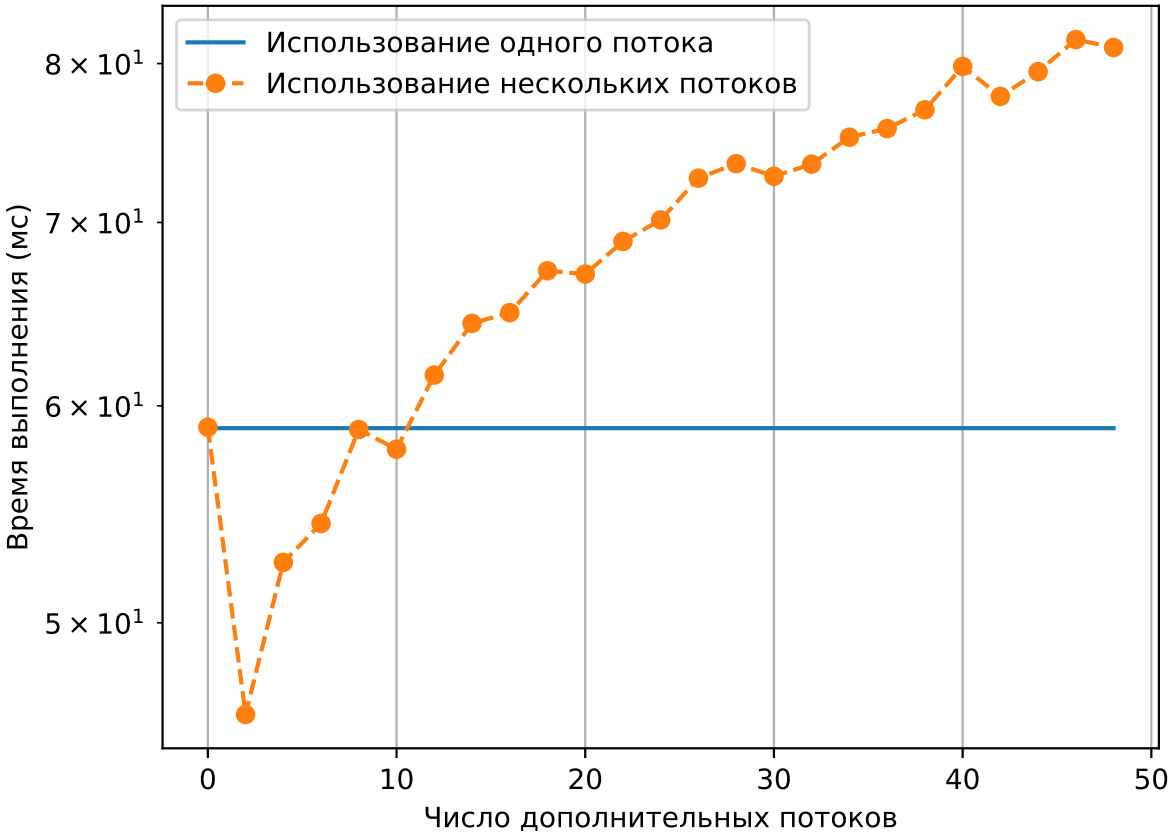

Рисунок 4.2 – Сравнение реализаций сортировок слиянием по времени с использованием логарифмической шкалы

Таблица 4.2 – Результаты замеров времени выполнения многопоточной реализации алгоритма сортировки слиянием для различного числа элементов массива, при 1 вспомогательном потоке

n	Один поток(мс)	Несколько потоков(мс)
10000	11.193	7.3873
20000	23.173	18.97
30000	33.781	28.963
40000	47.825	44.313
50000	58.886	54.198
60000	69.215	66.343
70000	83.156	80.055

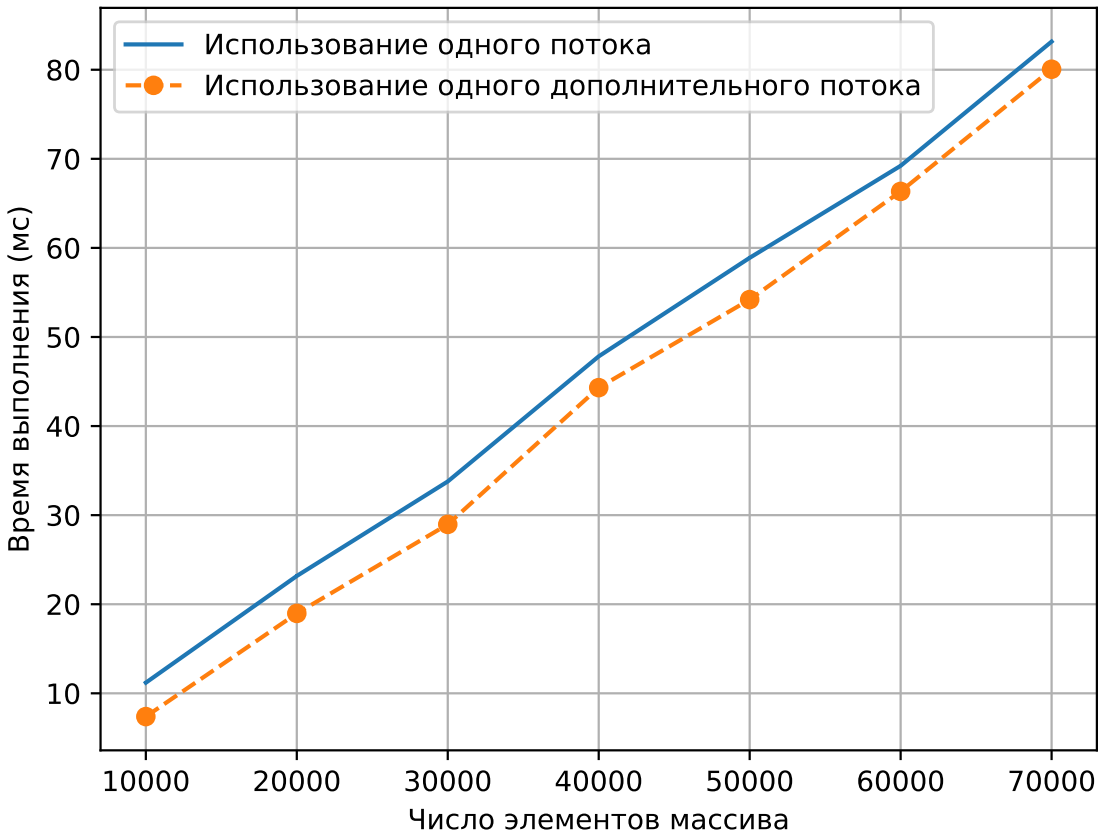

Рисунок 4.3 – Зависимость времени получения результата от числа элементов массива при различных реализациях сортировок

Вывод

В результате анализа таблицы 4.1 было получено, что при использовании 6 вспомогательных потоков при сортировке 50000 элементов требуется в 1.27 раз меньше времени для получения результата, чем при использовании одного потока.

Из таблицы 4.2 можно сделать выводы — при увеличении числа элементов в массиве время получения результата с использованием нескольких потоков также увеличивается. При увеличении числа сортируемых элементов с 10000 до 70000 время получения результата при использовании одного потока увеличилось в 7.4 раза.

При увеличении числа сортируемых элементов с 10000 до 70000 время получения результата при использовании 1 вспомогательного потока увеличилось в 10.8 раз.

Таким образом, рекомендуется использование 2 вспомогательных потоков, так как это позволит вспомогательным потокам сортировать подмассивы максимальной длины с минимальными тратами для переключения контекста.

ЗАКЛЮЧЕНИЕ

В результате исследования было получено, что при использовании 6 вспомогательных потоков при сортировке 50000 элементов требуется в 1.27 раз меньше времени для получения результата.

При увеличении числа элементов в массиве время получения результата с использованием нескольких потоков также увеличивается. При увлечении числа сортируемых элементов с 10000 до 70000 время получения результата при использовании одного потока увеличилось в 7.4 раза.

При увлечении числа сортируемых элементов с 10000 до 70000 время получения результата при использовании 1 вспомогательного потока увеличилось в 10.8 раз. Наилучший результат при сортировке 50000 был получен при использовании 2 потоков, так как в таком случае минимальное время тратится на переключение контекста и потоки сортируют подмассивы максимального размера.

Поставленная цель была достигнута: получены навыки организации параллельного выполнения операций.

Для поставленной цели были выполнены все поставленные задачи, а именно:

1) описан алгоритм сортировки слиянием;
2) разработаны версии приведенного алгоритма, при использовании одного и нескольких потоков;
3) определены средства программной реализации;
4) реализованы разработанные алгоритмы;
5) выполнены замеры процессорного времени работы различных реализаций алгоритма;
6) проведен сравнительный анализ зависимостей времени решения задачи от размерности входа и количества вспомогательных потоков.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Многопоточное программирование [Электронный ресурс]. — Режим доступа: https://studfile.net/preview/16573807/ (дата обращения: 26.12.2023).
2. Merge sort [Электронный ресурс]. — Режим доступа: https://nauchniestati.ru/spravka/algoritm-sortirovki-sliyaniem (дата обращения: 26.12.2023).
3. Процессорное время [Электронный ресурс]. — Режим доступа: https://prosto.1gb.ru/wiki/index.php?title=Р§СЃР«СЗРҳСҒСҒР«СЃР,,Р«Рҳ_РўСЃРҳРёCS (дата обращения: 26.12.2023).
4. clock(3) — Linux [Электронный ресурс]. — Режим доступа: https://man7.org/linux/man-pages/man3/clock.3.html (дата обращения: 28.12.2023).
5. std::thread [Электронный ресурс]. — Режим доступа: https://en.cppreference.com/w/cpp/thread/thread (дата обращения: 27.12.2023).

ПРИЛОЖЕНИЕ А

Листинг А.1 – Реализация алгоритма слияния отсортированных подмассивов

```
void merge(arrayT& arr, int left, int right) {

    int mid = (left + right) / 2;

    int i = left,
        j = mid + 1, pos = 0,
        len = right - left + 1;

    arrayT tmp(len);

    while (i <= mid && j <= right)
        tmp[pos++] = (arr[i] < arr[j]) ? arr[i++] : arr[j++];

    while (i <= mid)
        tmp[pos++] = arr[i++];

    while (j <= right)
        tmp[pos++] = arr[j++];

    for (int i = left; i <= right; ++i)
        arr[i] = tmp[i - left];
}
```

Листинг А.2 – Реализация алгоритма сортировки слиянием при использовании одного потока

```
void mergeSort(arrayT& arr, int left, int right) {

    if (right <= left)
        return;

    int mid = (right + left) / 2;

    mergeSort(arr, left, mid);
    mergeSort(arr, mid + 1, right);
    merge(arr, left, right);
}
```

Листинг А.3 – Реализация алгоритма сортировки слиянием при использовании заданного числа потоков

```
void mergeSortMultiThread(arrayT& arr, int left, int right, int
    threads) {

    if (left >= right)
        return;

    int mid = (right + left) / 2;

    if (threads == 1) {
        thread t(std::bind(mergeSortMultiThread, ref(arr), left,
            mid, 0));
        mergeSortMultiThread(arr, mid + 1, right, 0);
        t.join();
    }
    else if (threads >= 2) {
        int oddThread = threads % 2;
        thread t1(std::bind(mergeSortMultiThread, ref(arr),
            left, mid, threads / 2 + oddThread));
        thread t2(std::bind(mergeSortMultiThread, ref(arr), mid
            + 1, right, threads / 2));
        t1.join();
        t2.join();
    }
    else {
        mergeSortMultiThread(arr, left, mid, 0);
        mergeSortMultiThread(arr, mid + 1, right, 0);
    }

    merge(arr, left, right);
}
```