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 ИУ7-54Б (Группа) _____ (Подпись, дата) Разин А. (И. О. Фамилия)

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

2023 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, поскольку в случае многопоточности ресурсы одного или нескольких ядер используются несколькими потоками или процессами [1].

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. В случае многопроцессорных систем, включающих несколько физических процессоров, подход многопоточности направлен на максимальное использование ресурсов каждого ядра, позволяя осуществлять параллельную обработку на уровне потоков и инструкций. Иногда эти подходы объединяются в системах с несколькими многопоточными процессорами или ядрами для повышения общей производительности [1].

Целью данной лабораторной работы является изучение принципов и 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 1 потока и нескольких потоков.
3. Определить средства программной реализации.
4. Реализовать разработанные алгоритмы.
5. Выполнить замеры процессорного времени работы различных реализаций алгоритма.
6. Провести анализ времени получения отсортированных данных.

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 (англ. merge sort) является эффективным и стабильным алгоритмом сортировки, который применяет принцип «разделяй и властвуй» для упорядочивание элементов в массиве [2].

Алгоритм состоит из нескольких этапов [2].

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 сортировка: 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.

В результате выполнения этих шагов получается отсортированный массив, который содержит все элементы исходного массива.

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 составляет $O(n \cdot log(n))$ [2].

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. Возможно создание потока 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 предъявлены ряд требований:

– иметь интерфейс для выбора действий;

– динамически выделять память 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. В данной работе, при использовании многопоточности возможно ожидание одними потоками выполнения других потоков. Поток не выполняет никаких действий в это время, поэтому простой не влияет на процессорное время, однако это влияет на результирующее реальное время, поэтому 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

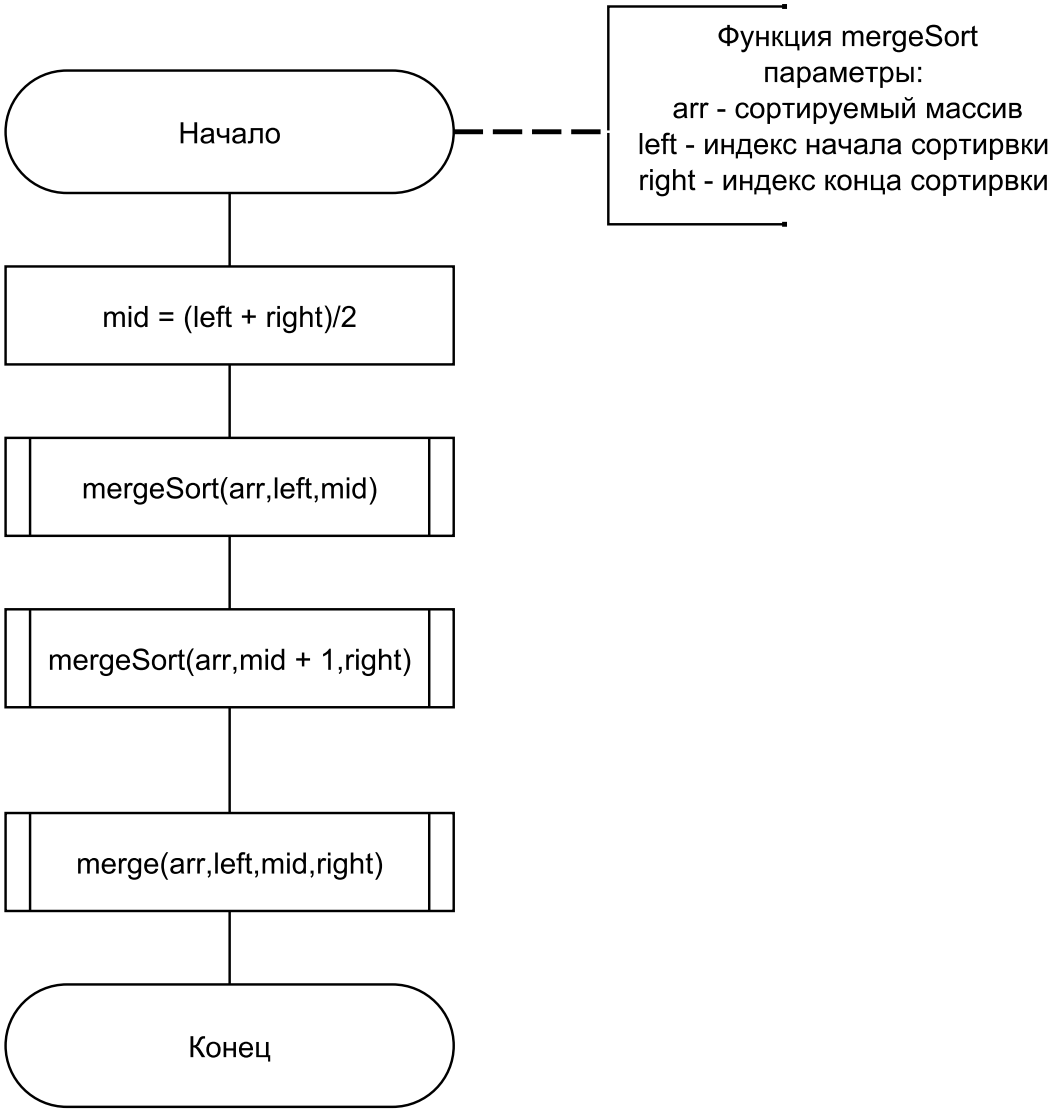

Рисунок 2.1 – Схема алгоритма сортировки слиянием, при использовании 1 потока

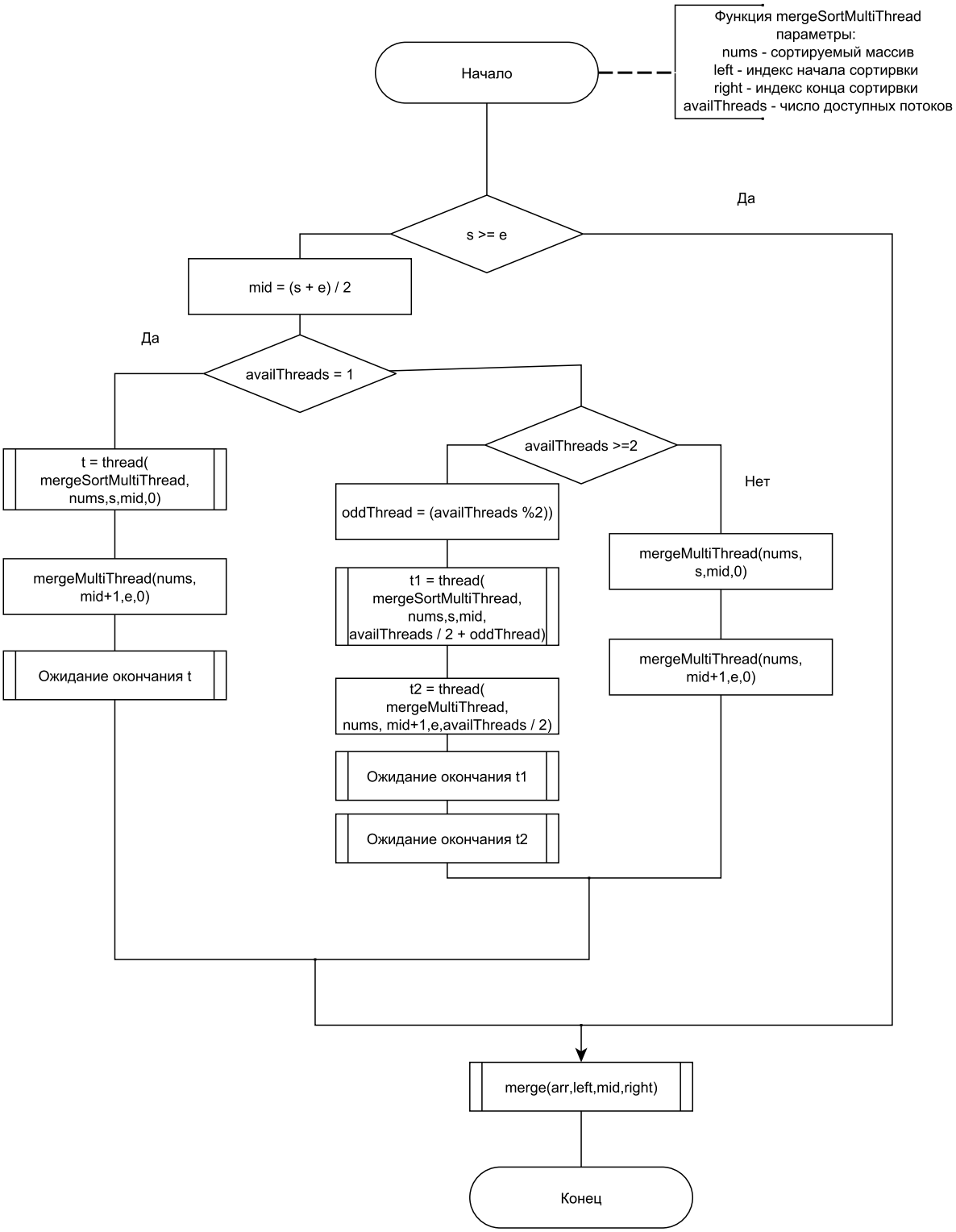

Рисунок 2.2 – Схема алгоритма сортировки слиянием, при использовании нескольких потоков

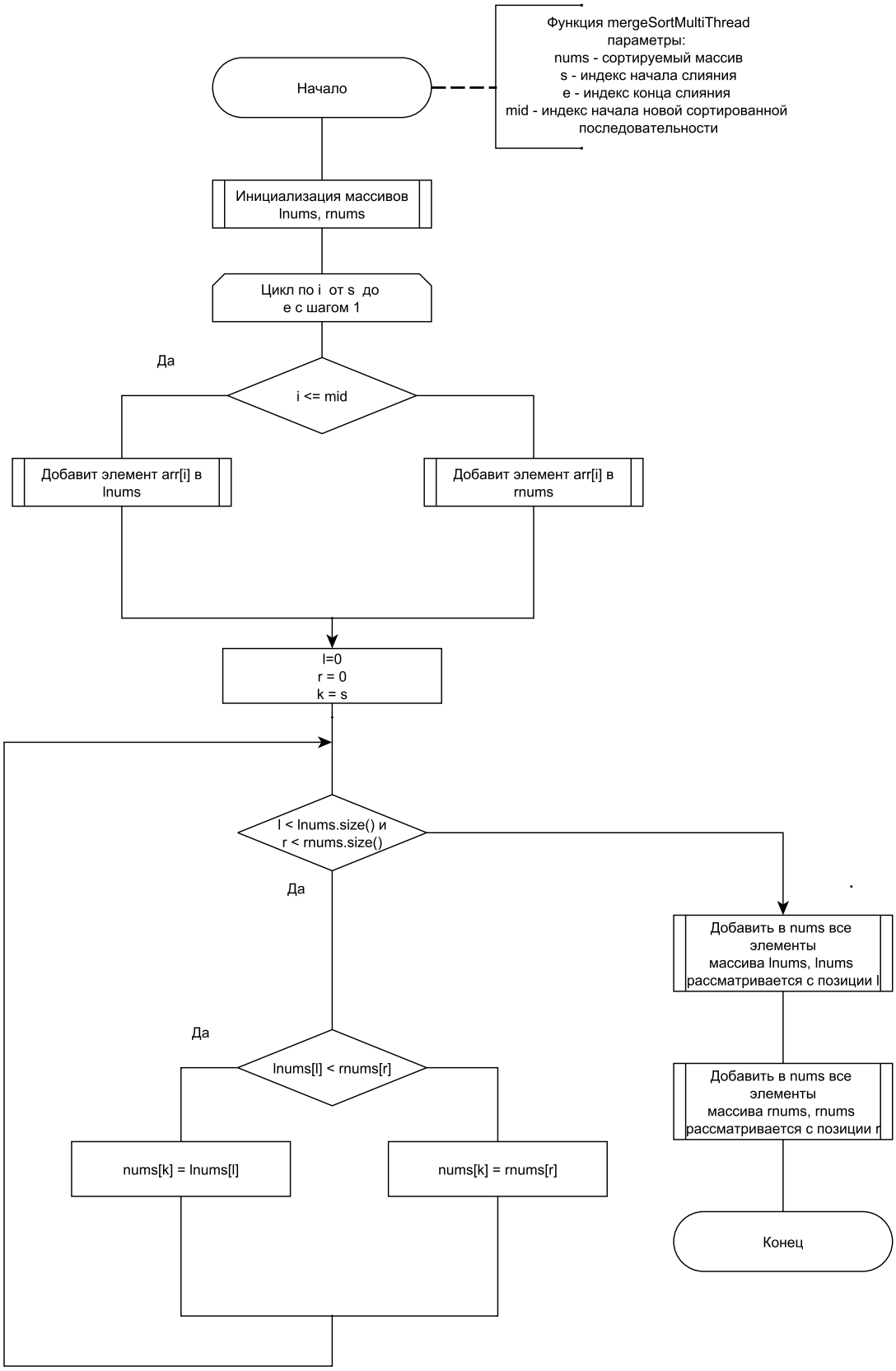

Рисунок 2.3 – Схема алгоритма слияния отсортированных последовательностей

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++, при использовании компилятора gcc версии 10.5.0, так как в стандартной 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 [3].

Для создания потоков и работы с ними был использован класс `thread` из стандартной библиотеки выбранного языка [4].

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;
}
```

В листинге 3.1, приведен работы с описанным классом, каждый объект класса представляет собой поток операционной системы, что позволяет нескольким функциям выполняться параллельно [4].

Поток начинает свою работу после создания объекта, соответствующего класса, запуская функцию, приведенную в его конструкторе [4]. В данном примере будет запущен 1 поток, который выполнит функцию `foo`, которая выведет число 10 на экран.

3.2 Реализация алгоритмов

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

3.3 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 91	-9 -3 1 2 9

Вывод

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

4 Исследовательская часть

В данном разделе будут приведены пример работы программы, условия исследования и сравнительный анализ алгоритмов на основе полученных данных.

4.1 Технические характеристики

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

1. Процессор Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz, 2592 МГц, ядер: 6, логических процессоров: 12.
2. Оперативная память: 16 ГБайт.
3. Операционная система: Майкрософт Windows 10 Pro [5].
4. Использованная подсистема: WSL2 [6].

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 приложения для алгоритмов сортировок.

```
Меню:
0)Выход
1)Сортировка массива с использованием сортировки слиянием при 1 потоке
2)Сортировка массива с использованием сортировки слиянием при нескольких потоках
3)Расчет времени

Введите пункт из меню: 2

Введите  размер массива для сортировки:
4
Введите массив для сортировки:
-4 1 100 -1
Введите число дополнительных потоков для сортировки
5
Отсортированный массив
-4 -1 1 100
```

Рисунок 4.1 – Пример работы программы

4.3 Временные характеристики

Результаты исследования временных замеров приведены в таблице 4.1.

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 n определяет число элементов в массиве, значения из столбца «Один поток» показывают результаты замеров работы реализации при использовании 1 потока, значение из столбца «Несколько потоков» показывают результаты замеров работы многопоточной реализации, результаты представлены в миллисекундах. «Число потоков» определяет число вспомогательных потоков, для получения результата при многопоточной реализации алгоритма. Результат приведен в миллисекундах

По таблице 4.1 был построен график 4.2.

Также была получена таблица 4.2, в которой в многопоточной реализации было использован 1 вспомогательный потока. По таблице 4.2 был получен график 4.3. Обозначения столбцов и условия получения данных аналогичны 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

Таблица 4.2 – Полученная таблица замеров по времени 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

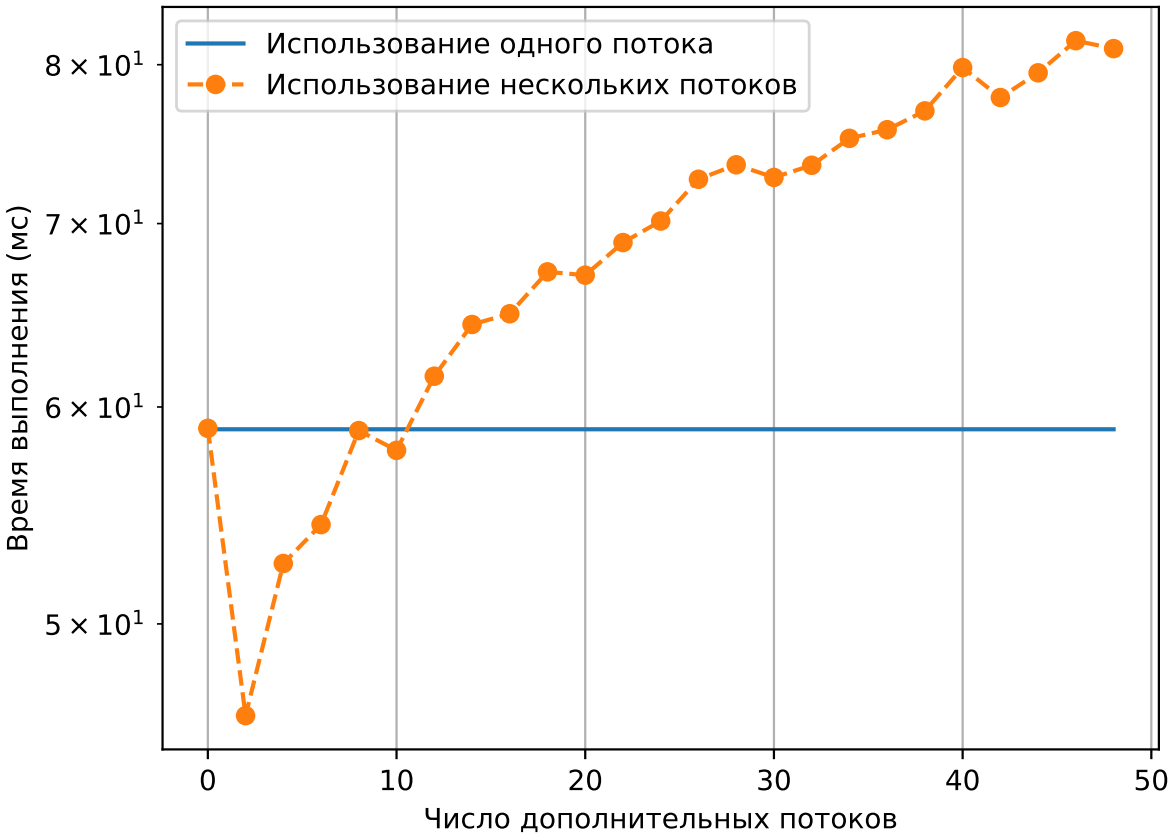

Рисунок 4.2 – Сравнение реализаций сортировок по времени с использованием логарифмической шкалы

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.

Из таблицы 4.2, можно сделать выводы, что 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.

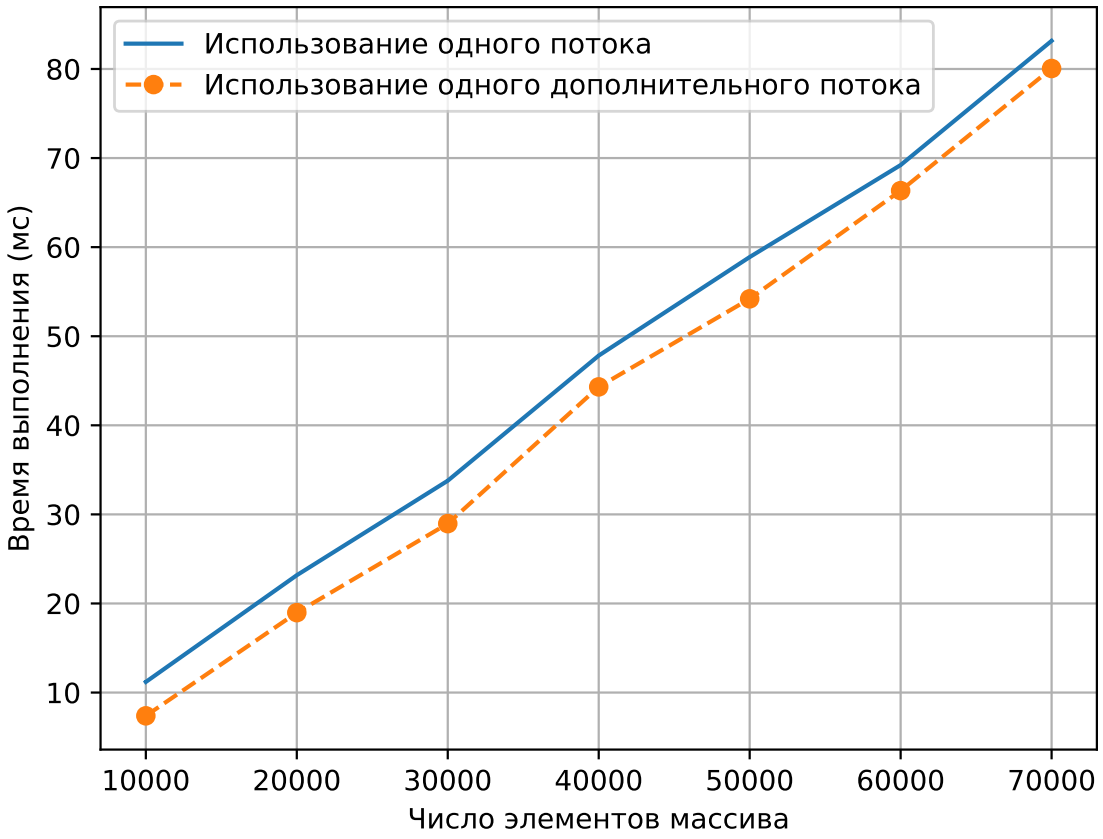

Рисунок 4.3 – Зависимость времени получения результата от числа элементов массива

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: изучены принципы и получены навыки организации параллельного выполнения операций.

Для поставленной цели были выполнены все поставленные задачи.

1. Описать алгоритм сортировки слиянием.
2. Разработать версии приведенного алгоритма, при использовании 1 потока и нескольких потоков.
3. Определить средства программной реализации.
4. Реализовать разработанные алгоритмы.
5. Выполнить замеры процессорного времени работы различных реализаций алгоритма.
6. Провести анализ времени получения отсортированных данных.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Многопоточное программирование [Электронный ресурс]. — Режим доступа: https://studfile.net/preview/16573807/ (дата обращения: 06.12.2023).
2. Merge sort [Электронный ресурс]. — Режим доступа: https://nauchniestati.ru/spravka/algoritm-sortirovki-sliyaniem (дата обращения: 06.12.2023).
3. clock(3) — Linux [Электронный ресурс]. — Режим доступа: https://man7.org/linux/man-pages/man3/clock.3.html (дата обращения: 28.09.2023).
4. std::thread [Электронный ресурс]. — Режим доступа: https://en.cppreference.com/w/cpp/thread/thread (дата обращения: 07.12.2023).
5. Windows 10 Pro 2h21 64-bit [Электронный ресурс]. — Режим доступа: https://www.microsoft.com/ru-ru/software-download/windows10 (дата обращения: 28.09.2023).
6. Что такое WSL [Электронный ресурс]. — Режим доступа: https://learn.microsoft.com/ru-ru/windows/wsl/about (дата обращения: 28.09.2023).

ПРИЛОЖЕНИЕ А

Листинг А.1 – Реализация слияния отсортированных подмассивов

```
void merge(std::vector<int>& nums, int s, int mid, int e)
{
    std::vector<int> lnums, rnums;
    for (int i = s; i <= e; i++)
    {
        if (i <= mid)
        {
            lnums.push_back(nums[i]);
        }
        else
        {
            rnums.push_back(nums[i]);
        }
    }

    int l = 0, r = 0, k = s;

    while (l < lnums.size() && r < rnums.size())
    {
        if (lnums[l] < rnums[r])
        {
            nums[k++] = lnums[l++];
        }
        else
        {
            nums[k++] = rnums[r++];
        }
    }

    while (l < lnums.size())
    {
        nums[k++] = lnums[l++];
    }
    while (r < rnums.size())
    {
        nums[k++] = rnums[r++];
    }
```

```
38 }
```

Листинг А.2 – Реализация алгоритма сортировки слиянием, при использовании 1 потока

```
1  void mergeSort(std::vector<int>& arr, int left, int right)
2  {
3      if (left >= right)
4      {
5          return;
6      }
7      int mid = (left + right) / 2;
8      mergeSort(arr, left, mid);
9      mergeSort(arr, mid + 1, right);
10
11     merge(arr, left, mid, right);
12 }
```

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

```
1  void mergeSortMultiThread(std::vector<int>& nums, int s, int e,
       int availThreads)
2  {
3      if (s >= e)
4      {
5          return;
6      }
7
8      int mid = (s + e) / 2;
9      if (availThreads == 1)
10     {
11         std::thread t(std::bind(mergeSortMultiThread,
               std::ref(nums), s, mid, 0));
12         mergeSortMultiThread(nums, mid + 1, e, 0);
13         t.join();
14     }
15     else if (availThreads >= 2)
16     {
17         int oddThread = (availThreads % 2);
18         std::thread t1(std::bind(mergeSortMultiThread,
               std::ref(nums), s, mid, availThreads / 2 +
               oddThread));
19         std::thread t2(std::bind(mergeSortMultiThread,
```

```
            std::ref(nums), mid + 1, e, availThreads / 2));
        t1.join();
        t2.join();
    }
    else
    {
        mergeSortMultiThread(nums, s, mid, 0);
        mergeSortMultiThread(nums, mid + 1, e, 0);
    }
    merge(nums, s, mid, e);
}
```