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 ИУ «Информатика и системы управления»

КАФЕДРА ИУ-7 «Программное обеспечение ЭВМ и информационные технологии»

ОТЧЕТ

по лабораторной работе № 4

по курсу «Анализ алгоритмов»

на тему: «Параллельные вычисления на основе нативных потоков»

Студент ИУ7-55Б _____ И. Д. Половинкин
(Подпись, дата)

Преподаватель _____ Л. Л. Волкова
(Подпись, дата)

2024 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ **3**

1 Аналитический раздел **4**
1.1 Сортировка слиянием 4
1.1.1 Последовательная версия 4
1.1.2 Параллельная версия 5
1.2 Использование средств синхронизации 5

2 Конструкторский раздел **7**
2.1 Алгоритмы сортировки слиянием 7
2.1.1 Последовательная версия 7
2.1.2 Параллельная версия 8

3 Технологический раздел **11**
3.1 Требования к программному обеспечению 11
3.2 Средства реализации 11
3.3 Модули программы 11
3.3.1 Последовательная сортировка слиянием 11
3.3.2 Параллельная сортировка слиянием 13
3.4 Тестирование 14

4 Исследовательский раздел **16**
4.1 Технические характеристики 16
4.2 Последовательное выполнение 17
4.3 Последовательное и параллельное выполнение алгоритма 18

ЗАКЛЮЧЕНИЕ **20**

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ **21**

ВВЕДЕНИЕ

При необходимости максимально эффективно использовать ресурсы системы для выполнения множества задач, важно стремиться к увеличению скорости работы программ. Недоступность возможности увеличения тактовой частоты процессора по определённым техническим причинам не означает, что нет альтернативного способа повышения производительности. Один из таких способов заключается в использовании многоядерных процессоров, что требует особого подхода к программированию.

Параллельное программирование – подход в технологии разработки программного обеспечения, которые основывается на потоках [1]. Поток – часть кода программы, которая может выполняться параллельно с другими частями кода программы. Многопоточность – способность центрального процессора или одного ядра в многоядерном процессоре одновременно выполнять несколько потоков.

Целью лабораторной работы является изучение и реализация параллельного программирования для сортировки массива методом слияния. Для её достижения необходимо выполнить следующие задачи:

- рассмотреть случай последовательного и параллельного алгоритма сортировки слиянием;
- привести схемы алгоритмов последовательной и параллельной сортировки слиянием;
- реализовать приведенные алгоритмы;
- выполнить тестирование реализации алгоритмов методом черного ящика;
- провести сравнительный анализ этих алгоритмов по выполнению времени на основе экспериментальных данных.

1 Аналитический раздел

В данном разделе рассматривается базовый алгоритм сортировки слиянием, а также идея его параллельной версии.

1.1 Сортировка слиянием

1.1.1 Последовательная версия

Сортировка слиянием определяется как алгоритм сортировки, который работает путем разделения массива на более мелкие подмассивы, сортировки каждого подмассива, а затем обратного слияния отсортированных подмассивов для формирования окончательного отсортированного массива [2].

Пример сортировки представлен на рисунке 1.1.

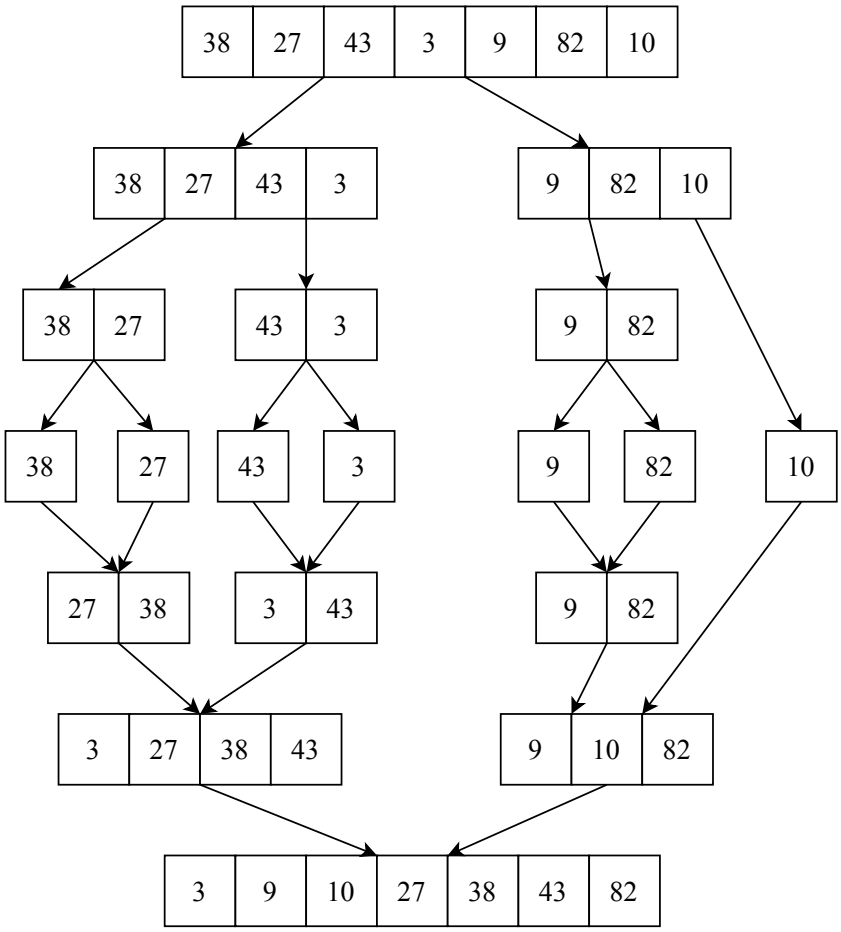

Рисунок 1.1 – Пример сортировки слиянием

1.1.2 Параллельная версия

Идея параллельного выполнения сортировки слиянием представлена на рисунке 1.2.

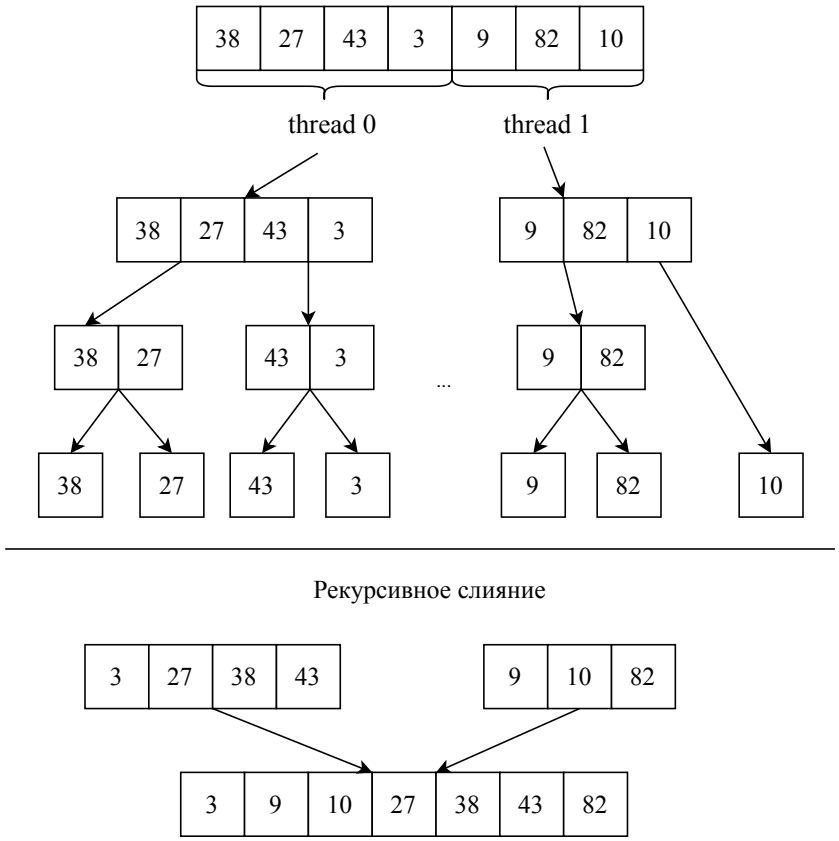

Рисунок 1.2 – Идея параллельного выполнения сортировки слиянием

Пусть есть два потока thread 0 и thread 1. Разделим исходный массив на две части таким образом, что одна часть будет обработана потоком thread 0, а другая – потоком thread 1 [3]. После этого каждый из потоков будет рекурсивно применять алгоритм сортировки слиянием к своей части массива. На выходе из этого шага получится два отсортированных локально фрагмента массива. Затем эти фрагменты объединяются для формирования глобально отсортированного массива. На нижней части рисунка 1.2 показано, что для получения глобально отсортированного массива требуется выполнить рекурсивное слияние.

1.2 Использование средств синхронизации

В таблице 1.1 приведена сравнительная таблица необходимости использования средств синхронизации.

Таблица 1.1 – Сравнение средств синхронизации

Средство синхронизации	**Описание**	**Необходимость использования**
Мьютекс	Переменная, блокирующая доступ к разделяемым ресурсам. Только поток-владелец мьютекса, захвативший его, может его освободить.	-
Семафоры	Неотрицательная защищенная переменнная, на которой определены операции: захватить P и освободить V. Семафор может захватить один процесс, а освободить совершенно другой.	-

Использование таких средств синхронизации как мьютекс или семафор не требуется для распараллеливания алгоритма сортировки слиянием. Требуется лишь ожидание завершение всех выделенных потоков процесса кроме главного, а после выполнить операцию слияния результата.

Вывод

В разделе была рассмотрена последовательная версия сортировки слиянием, предложена параллельная версия за счет разделения исходного массива на части, которые будут обрабатывать отдельные потоки. В качестве средства синхронизации выбрано ожидание завершения всех потоков процесса кроме главного.

2 Конструкторский раздел

В данном разделе приводятся схемы последовательного и параллельного алгоритмов сортировки слиянием, а также обоснование необходимости объединения промежуточных значений.

2.1 Алгоритмы сортировки слиянием

2.1.1 Последовательная версия

Алгоритм последовательной сортировки слиянием представлен на рисунке 2.1.

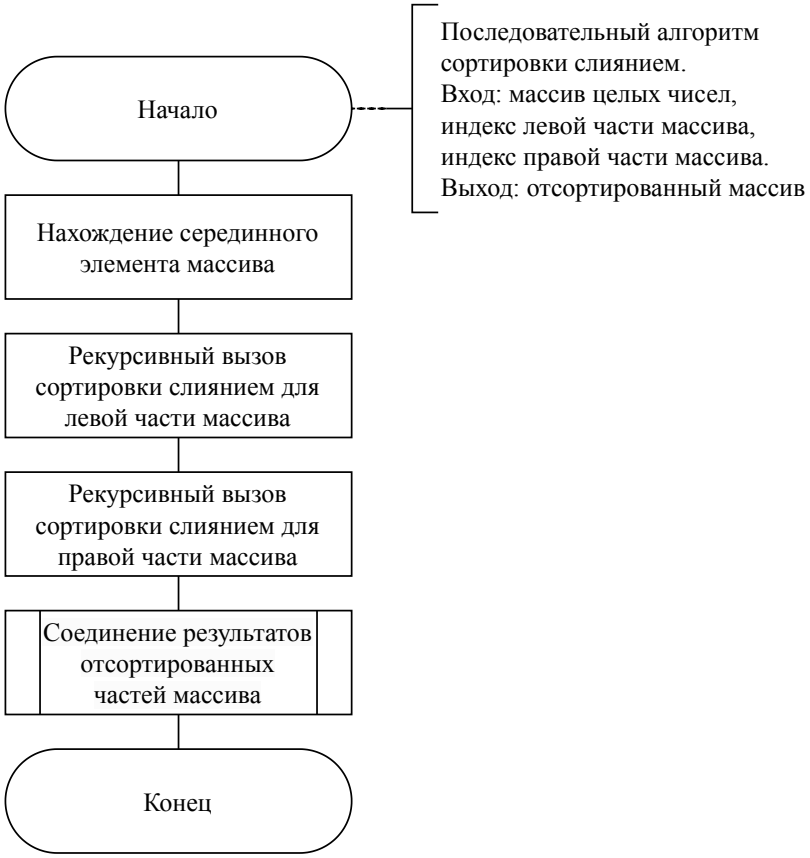

Рисунок 2.1 – Алгоритм последовательной сортировки слиянием

Алгоритм слияния результатов представлен на рисунке 2.2.

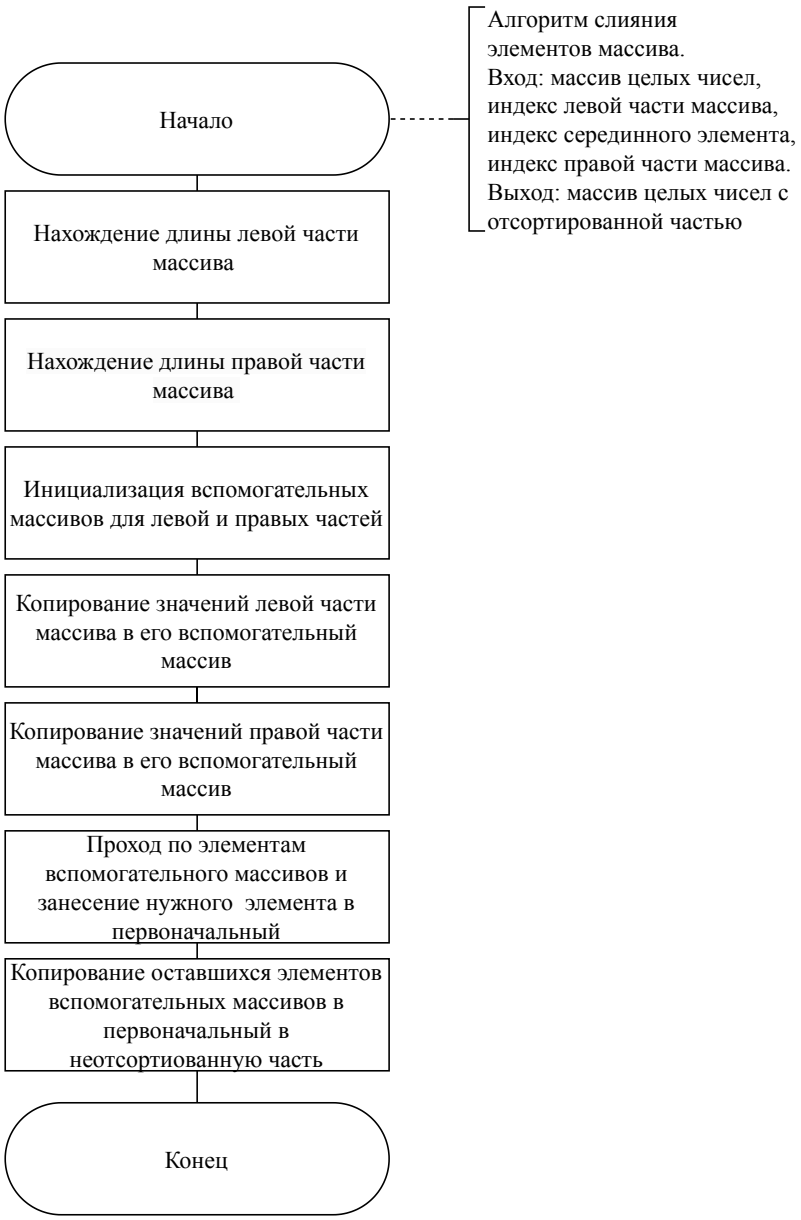

Рисунок 2.2 – Алгоритм слияния результатов

2.1.2 Параллельная версия

Алгоритм параллельной сортировки слиянием представлен на рисунке 2.3.

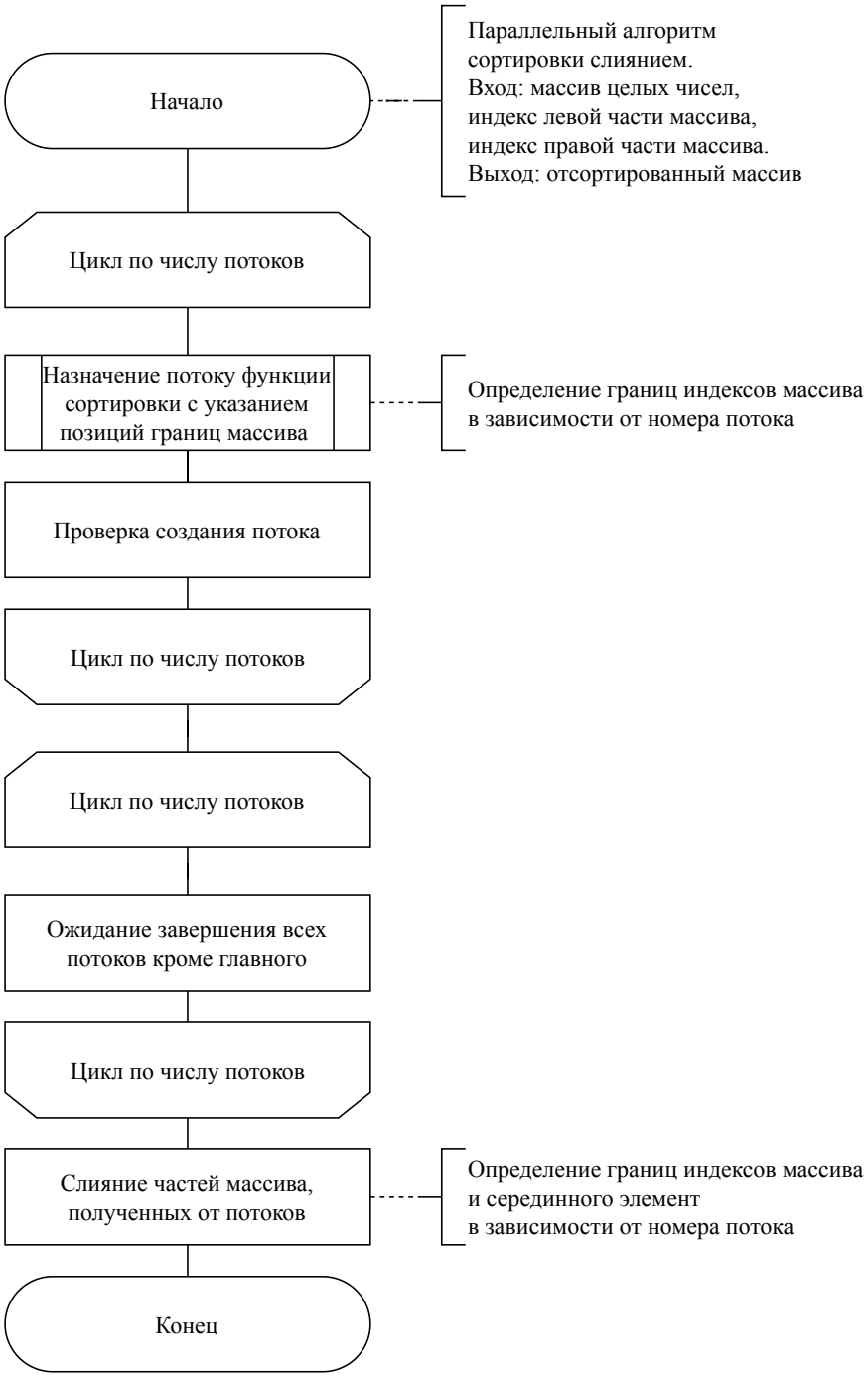

Рисунок 2.3 – Алгоритм параллельной сортировки слиянием

Главный поток выполняет создание вспомогательных потоков, назначение каждому из нему задания и запуск этого потока. После выполняется шаг объединения промежуточных значений, которые затем сливаются в единый отсортированный массив главным потоком. Средства синхронизации не требуются.

Алгоритм работы вспомогательного потока представлен на рисунке 2.4.

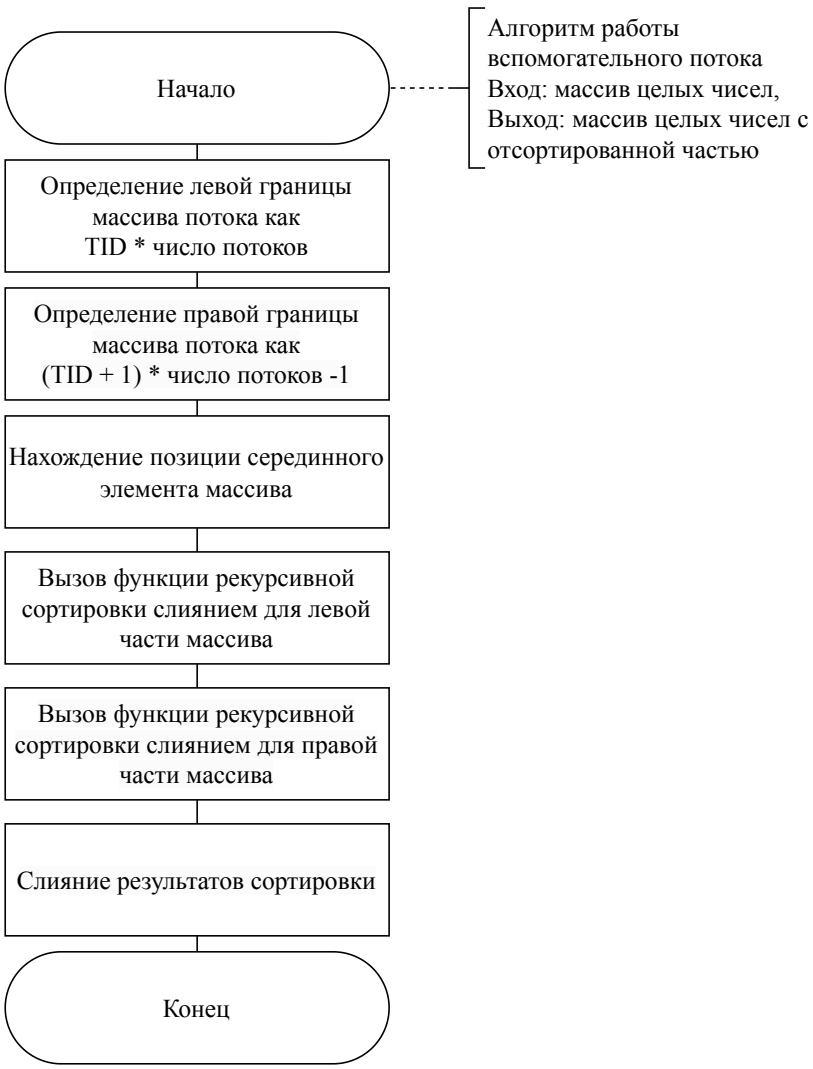

Рисунок 2.4 – Алгоритм работы вспомогательного потока

Вывод

В данном разделе были приведены схемы последовательного и параллельного алгоритмов сортировки слиянием. Средства синхронизации не требуются, необходимо использовать объединение результатов вспомогательных потоков.

3 Технологический раздел

В данном разделе рассматриваются требования к программному обеспечению, средства реализации, а также приводятся листинги алгоритмов последовательной и параллельной сортировки слиянием.

3.1 Требования к программному обеспечению

Программа должна удовлетворять ряду требований:

1. В основе распараллеливания лежат нативные потоки.
2. На вход алгоритму сортировки подается сгенерированный массив целых чисел.

3.2 Средства реализации

Для реализации алгоритмов был выбран язык программирования C [4]. В нем присутствуют нативные потоки библиотеки pthread [5], которая предоставляет интерфейс для создания потоков pthread_create(), ожидание завершения всех потоков pthread_join(). Замер времени расчета используется из библиотеки sys/time [6] функцией gettimeofday().

3.3 Модули программы

3.3.1 Последовательная сортировка слиянием

Реализация последовательной сортировки слиянием приведена на листинге 3.1.

Листинг 3.1 – Последовательная сортировка слиянием

```
void merge_sort(int arr[], int left, int right) {
    if (left < right) {
        int middle = left + (right - left) / 2;
        merge_sort(arr, left, middle);
        merge_sort(arr, middle + 1, right);
        merge(arr, left, middle, right);
    }
}
```

Функция слияния приведена на листинге 3.2.

Листинг 3.2 – Функция слияния

```
void merge(int arr[], int left, int middle, int right) {
    int i = 0, j = 0, k = 0;
    int left_length = middle - left + 1;
    int right_length = right - middle;
    int left_array[left_length], right_array[right_length];

    // копирование значений левой части массива
    // в вспомогательный
    for (int i = 0; i < left_length; i ++) {
        left_array[i] = arr[left + i];
    }

    // копирование значений правой части массива
    // в вспомогательный
    for (int j = 0; j < right_length; j ++) {
        right_array[j] = arr[middle + 1 + j];
    }

    i = 0, j = 0;
    // выбор элементов из вспомогательных
    // массивов для сортировки
    while (i < left_length && j < right_length) {
        if (left_array[i] <= right_array[j]) {
            arr[left + k] = left_array[i];
            i++;
        } else {
            arr[left + k] = right_array[j];
            j++;
        }
        k++;
    }

    // копирование оставшихся элементов из вспомогательных
    // массивов в основной
    while (i < left_length) {
        arr[left + k] = left_array[i];
        k++;
        i++;
    }
    while (j < right_length) {
        arr[left + k] = right_array[j];
        k++;
        j++;
    }
}
```

3.3.2 Параллельная сортировка слиянием

Создание главным потоком вспомогательных и назначение им заданий приведено на листинге 3.3.

Листинг 3.3 – Работа главного потока

```
void parallel_merge_sort() {
    struct timeval  start, end;
    double time_spent;

    pthread_t threads[NUM_THREADS];

    // Назначение каждому потоку задание -- сортировка
    for (long i = 0; i < NUM_THREADS; i ++) {
        int rc = pthread_create(&threads[i], NULL,
            thread_merge_sort, (void *)i);
        if (rc){
            printf("ERROR; return code from pthread_create() is %d\n", rc);
            exit(-1);
        }
    }

    // Ожидание завершения всех потоков
    for(long i = 0; i < NUM_THREADS; i++) {
        pthread_join(threads[i], NULL);
    }

    // Слияние результатов потоков
    merge_sections_of_array(arr, NUM_THREADS, 1);
}
```

Функция-задание для работы вспомогательного потока представлена на листинге 3.4.

Листинг 3.4 – Задание для работы вспомогательного потока

```
void *thread_merge_sort(void* arg)
{
    int thread_id = (long)arg;
    int left = thread_id * (NUMBERS_PER_THREAD);
    int right = (thread_id + 1) * (NUMBERS_PER_THREAD) - 1;
    if (thread_id == NUM_THREADS - 1) {
        right += OFFSET;
    }
    int middle = left + (right - left) / 2;
    if (left < right) {
        merge_sort(arr, left, right);
        merge_sort(arr, left + 1, right);
        merge(arr, left, middle, right);
    }
    return NULL;
}
```

Функция слияния полученных от потоков результатов представлена на листинге 3.5.

Листинг 3.5 – Слияние полученных от потоков результатов

```
void merge_sections_of_array(int arr[], int number, int aggregation) {
    for(int i = 0; i < number; i = i + 2) {
        int left = i * (NUMBERS_PER_THREAD * aggregation);
        int right = ((i + 2) * NUMBERS_PER_THREAD * aggregation) - 1;
        int middle = left + (NUMBERS_PER_THREAD * aggregation) - 1;
        if (right >= LENGTH) {
            right = LENGTH - 1;
        }
        merge(arr, left, middle, right);
    }
    if (number / 2 >= 1) {
        merge_sections_of_array(arr, number / 2, aggregation * 2);
    }
}
```

3.4 Тестирование

Для тестирования используется метод черного ящика. В данном разделе приведена таблица 3.1, в которой указаны классы эквивалентностей тестов.

Таблица 3.1 – Таблица тестов

№	Описание теста	Вход	Ожидаемый выход
1	Массив длиной 10 (случайный порядок)	1241 1302 7480 3885 9857 5624 3906 9271 982 6372	982 1241 1302 3885 3906 5624 6372 7480 9271 9857
2	Массив длиной 1	1	1
3	Массив длиной 2 (случайный порядок)	7480 3885	3885 7480

Для последовательной и параллельной реализации алгоритма сортировки слиянием все тесты пройдены успешно.

Вывод

В данном разделе был обоснован выбор языка программирования, используемых функций библиотек. Реализованы функции, описанные в разделах 1 и 2, проведено их тестирование методом черного ящика по таблице 3.1.

4 Исследовательский раздел

4.1 Технические характеристики

Технические характеристии устройства, на котором выполнялось тестирование:

- операционная система: Ubuntu 20.04;
- память: 8 GiB;
- процессор: Intel(R) Core(TM) i5-8265U CPU @ 1.60GHz 1.80 GHz;
- количество ядер: 4;
- количество логических ядер: 8.

Тестирование проводилось на ноутбуке, который был подключен к сети питания. Во время проведения тестирования ноутбук был нагружен только встроенными приложениями окружения, самим окружением и системой тестирования.

4.2 Последовательное выполнение

Проведем анализ сравнения времени расчета реализации последовательного алгоритма от длины массива для $N \in \{1000, 2000, 3000, 4000, ..., 10000\}$. Проведем работу каждого из них n = 100 раз и поделим на число n. Получим среднее значение работы каждого из алгоритмов.

На рисунке 4.1 приведена зависимость времени работы последовательного алгоритма сортировки слиянием от размера массива.

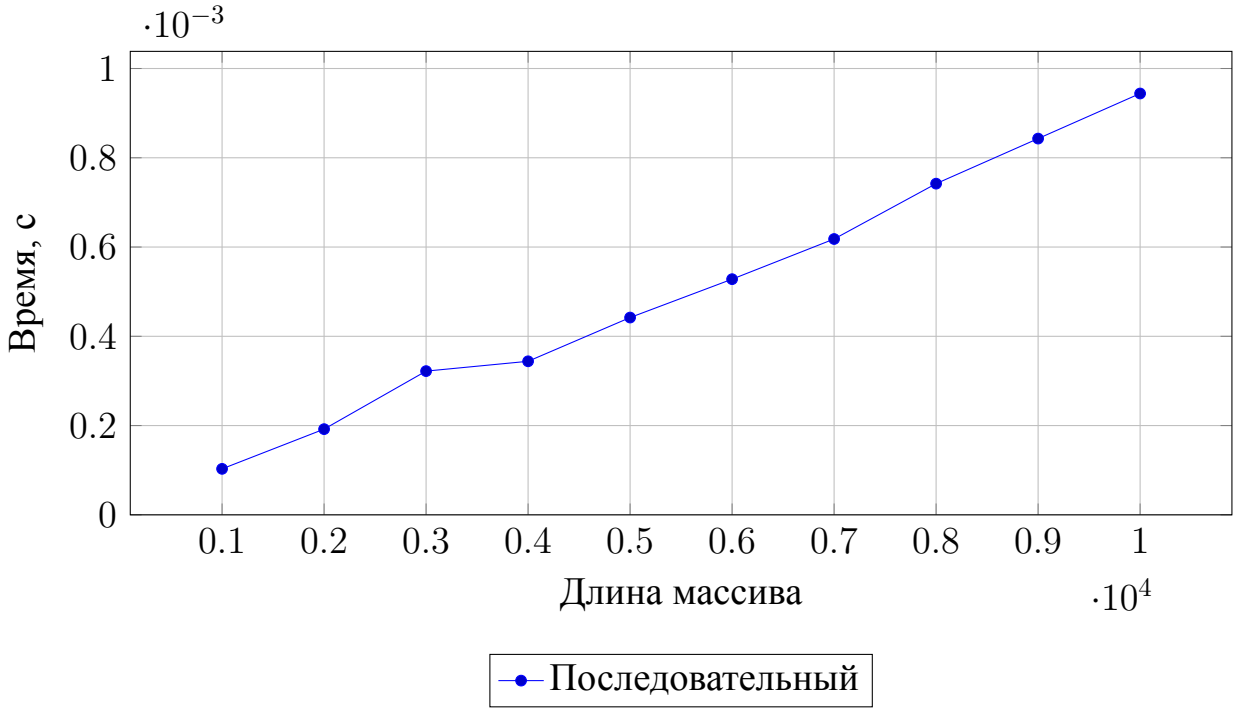

Рисунок 4.1 – Зависимость времени работы последовательного алгоритма сортировки слиянием от размера массива

4.3 Последовательное и параллельное выполнение алгоритма

Проведем анализ сравнения времени расчета реализаций последовательного алгоритма и параллельного с одним вспомогательным потоком для длины массива для $N \in \{1000, 2000, 3000, 4000, ..., 10000\}$. Проведем работу каждого из них n = 100 раз и поделим на число n. Получим среднее значение работы каждого из алгоритмов.

На рисунке 4.2 приведены зависимости времени работы версии алгоритмов от размеров массива.

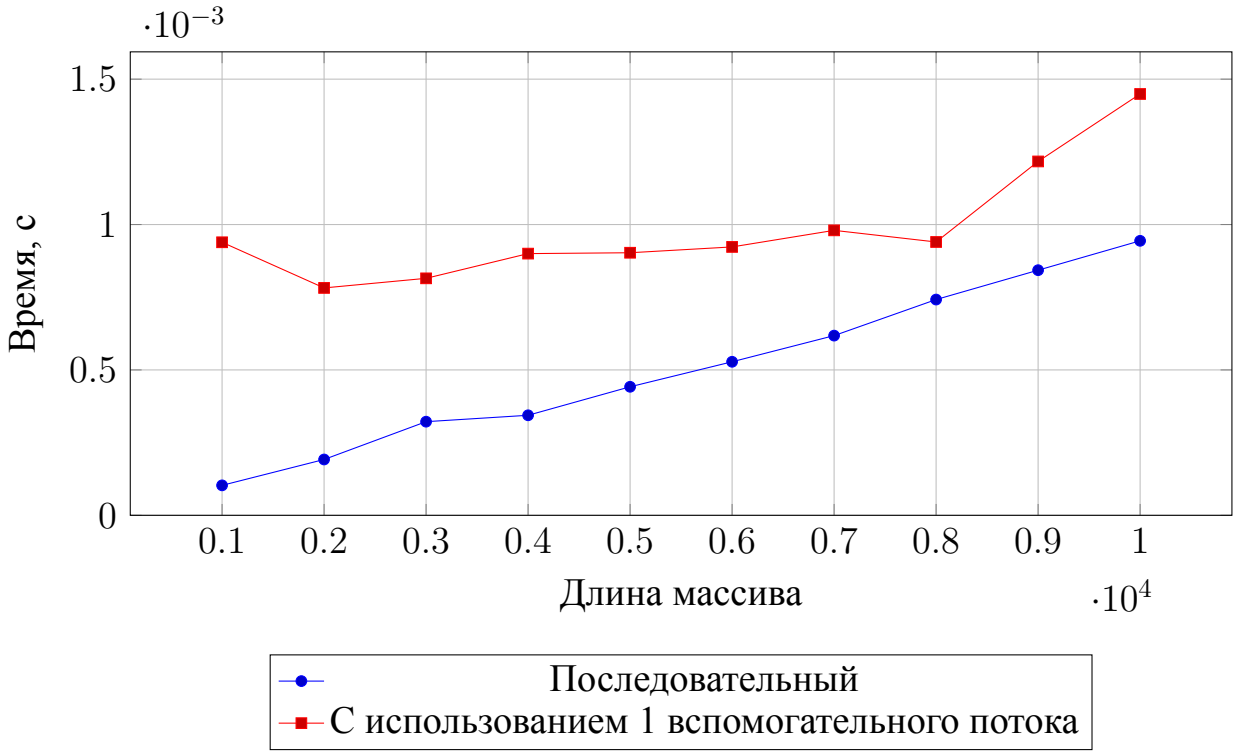

Рисунок 4.2 – Зависимости времени работы версии алгоритмов от размеров массива

Проведем анализ времени работы алгоритма для последовательной и параллельной реализации алгоритмов сортировки слиянием. Проведем работу каждого из них n = 100 раз и поделим на число n. Получим среднее значение работы каждого из алгоритмов. Для параллельного алгоритма выполнение вычислений производилось на $K \in \{1, 2, 4, 8, 16, ..., 8 \cdot Q\}$ потоках, где Q – число логических ядер представленной ЭВМ.

На рисунке 4.3 приведена зависимость времени работы параллельного алгоритма сортировки слиянием от числа используемых потоков.

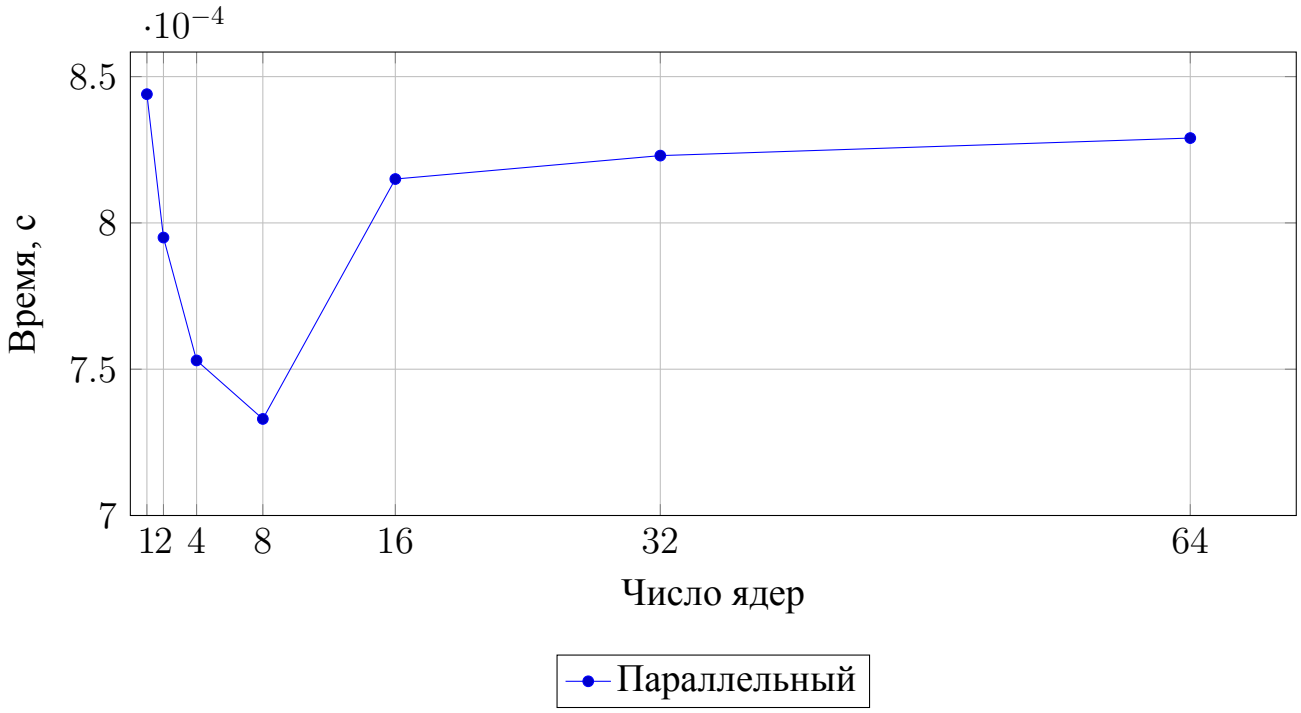

Рисунок 4.3 – Зависимость времени работы параллельного алгоритма сортировки слиянием от числа используемых потоков

Вывод

Использование одного вспомогательного потока не является целесообразным решением, поскольку необходимо тратить дополнительные операции по его созданию, выделению ему ресурсов, а также его уничтожения. В результате эксперимента было получено, что при использовании 8 потоков время многопоточной реализации алгоритма сортировки слиянием меньше остальных рассмотренных случаев.

ЗАКЛЮЧЕНИЕ

В ходе выполнения лабораторной работы были рассмотрены последовательная и параллельная реализация алгоритма сортировки слиянием. Были выполнено описание каждого из этих алгоритмов, приведены соответствующие схемы алгоритмов.

Многопоточная версия алгоритма демонстрирует ускорение в среднем в 2.5 раза по сравнению с последовательной реализацией. Однако использование одного дополнительного потока не является эффективным решением в качестве замены последовательной версии из-за затрат на его создание, выделение ресурсов и уничтожение, что приводит к приросту времени выполнения в среднем в 1.5-2 раза. Оптимальным выбором оказывается использование 8 потоков, что минимизирует время работы алгоритма сортировки слиянием по сравнению с другими вариантами. Для наилучшего распараллеливания рекомендуется использовать количество потоков, соответствующее числу логических ядер устройства.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Таненбаум, Э., Бос Х. Современные операционные системы. 4-е изд. / Э. Таненбаум, Х. Бос. — СПб.: Питер, 2015. - 1120 с.: ил. - (Серия ”Классика computer science”)

2. Merge Sort - Data Structure and Algorithms Tutorials - электронная версия [Электронный ресурс]. – Режим доступа, URL: https://www.geeksforgeeks.org/merge-sort/ (дата обращения: 21.02.2024)

3. Parallel Merge Sort with Pthreads - электронная версия [Электронный ресурс]. – Режим доступа, URL: https://malithjayaweera.com/2019/02/parallel-merge-sort/ (дата обращения: 21.02.2024)

4. C language - электронная версия [Электронный ресурс]. – Режим доступа, URL: https://en.cppreference.com/w/c/language (дата обращения: 21.02.2024)

5. pthreads - Linux manual page - электронная версия [Электронный ресурс]. – Режим доступа, URL: https://man7.org/linux/man-pages/man7/pthreads.7.html (дата обращения: 21.02.2024)

6. gettimeofday - Linux manual page [Электронный ресурс]. – Режим доступа, URL: https://man7.org/linux/man-pages/man2/gettimeofday.2.html (дата обращения: 05.02.2024)