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

Отчёт по лабораторной работе №4 по дисциплине "Анализ алгоритмов"

Тема Параллельный алгоритм сортировки слиянием

Студент Жабин Д.В.

Группа ИУ7-54Б

Преподаватель Волкова Л.Л.

Москва, 2021 г.

Оглавление

Введение

Современная архитектура электронно-вычислительных машин позволяет работать с параллелизмом. Параллелизм — это свойство систем, при котором несколько вычислений выполняются одновременно и при этом, возможно, взаимодействуют друг с другом. Вычисления могут выполняться на нескольких ядрах одного чипа с вытесняющим разделением времени потоков на одном процессоре, либо выполняться на физически отдельных процессорах. Для выполнения параллельных вычислений разработан ряд математических моделей, в том числе сети Петри, исчисление процессов, модели параллельных случайных доступов к вычислениям и модели акторов.

Многие языки программирования дают возможность работать с потоками. Потоки — абстракция операционной системы, позволяющая выполнять части кода параллельно. Каждый поток имеет `контекст`, в который входит участок памяти под стек и копию регистров процессора. Потоков в системе может быть запущено куда больше, чем имеется процессоров и ядер, поэтому операционная система занимается диспетчеризацией и планированием выполнения процессов, восстанавливая регистры в ядрах процессора для работы некоторого потока некоторое время, затем сохраняет обратно и даёт выполняться следующему.

Потоки можно погружать в сон на некоторое время, а также разделять один и тот же участок памяти компьютера между потоками.

Так, работа с потоками позволит в разы увеличить производительность того или иного алгоритма.

Целью лабораторной работы является исследование возможностей параллельных вычислений в системе. Для её достижения поставлены следующие задачи:

- выбрать стандартный алгоритм, производящий вычисления или работающий с памятью;
- изучить его и оценить возможность распараллеливания с целью увеличения производительности;
- разработать схемы обоих алгоритмов;

- реализовать изученные алгоритмы;
- провести тестирование реализаций алгоритмов;
- провести сравнительный анализ скорости работы реализаций алгоритмов с учетом возможности выделения разного количества потоков.

1 Аналитическая часть

Рассмотрим стандартный алгоритм сортировки массива слиянием, который в дальнейшем будет распараллелен.

1.1 Стандартный алгоритм сортировки слиянием

Стандартный алгоритм сортировки слиянием [4] описывает стратегию `Разделяй и властвуй`.

При использовании этой стратегии основная задача делится на подзадачи. Когда решение для каждой подзадачи готово, их результаты объединяются для решения основной задачи.

Пусть, необходимо отсортировать массив $array$. Подзадача состоит в том, чтобы отсортировать часть этого массива, начиная с индекса $begin$ и заканчивая индексом end, обозначенную как $array[begin..end]$.

Разделяй

Если $point$ является промежуточной точкой между $begin$ и end, то можно разбить массив $array[begin..end]$ на два подмассива $subarray[begin..point]$ и $subarray[point + 1..end]$.

Влавствуй

На это этапе происходит попытка отсортировать оба подмассива $subarray[begin..point]$ и $subarray[point + 1..end]$. Если еще не достигнут базовый вариант (размер подмассива меньше 2), можно снова разбить каждый подмассив и попытаться отсортировать их.

Комбинируй

Когда достигается базовый вариант, получается два отсортированных подмассива $subarray[begin..point]$ и $subarray[point + 1..end]$ для массива $array[begin..end]$. Результаты объединяются, создавая отсортированный массив $array[begin..end]$ из двух отсортированных подмассивов.

Стандартный алгоритм сортировки слиянием реализует вышеописанную стратегию `Разделяй и властвуй`. На рисунке (1.1) приведен пример работы этого алгоритма.

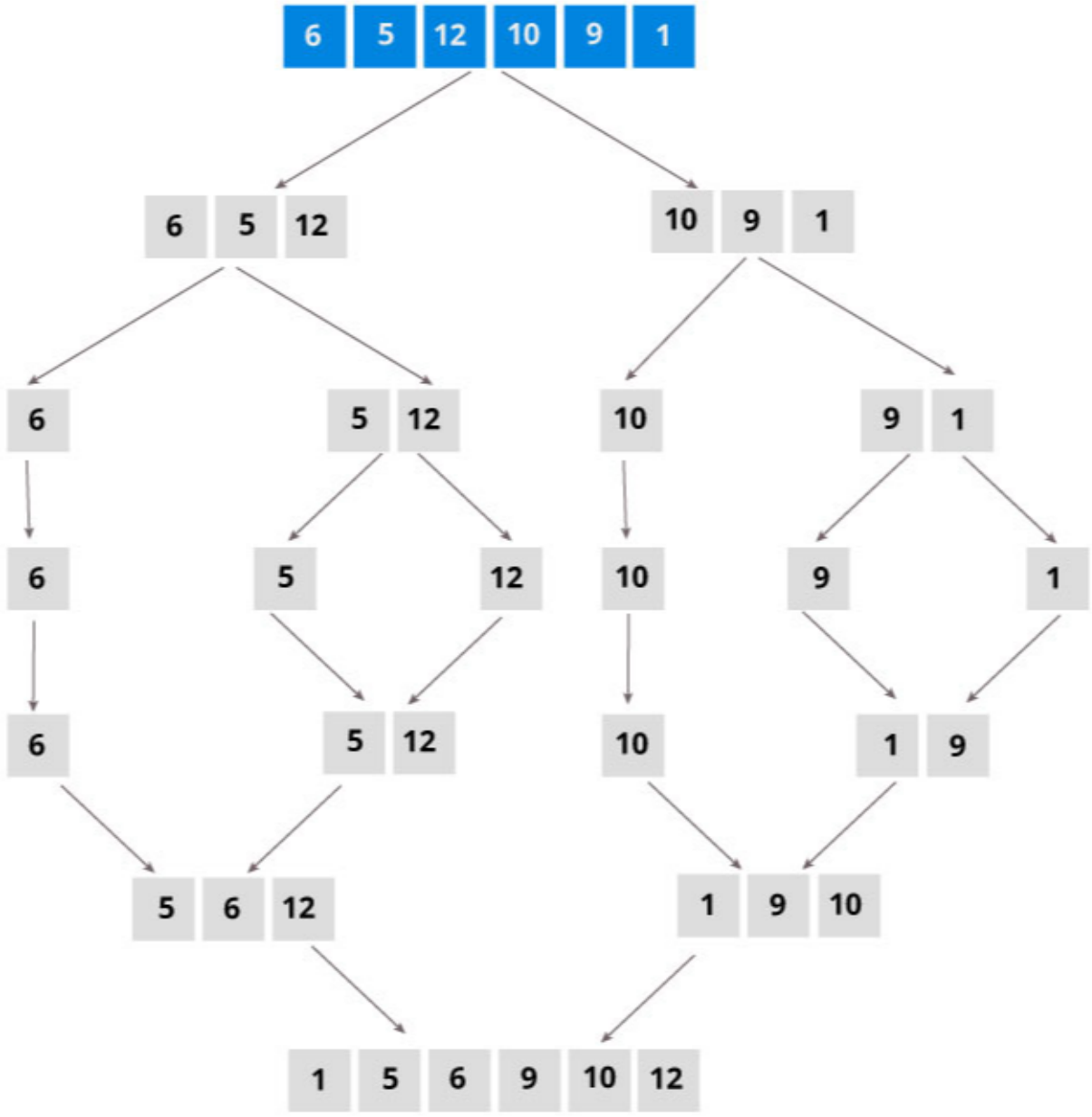


Рисунок 1.1 — Пример работы стандартного алгоритма сортировки слиянием

1.2 Вывод по аналитической части

В данном разделе был рассмотрен стандартный алгоритм сортировки слиянием. Можно заметить, что алгоритм допускает естественное распараллеливание.

2 Конструкторская часть

На основе полученных аналитических данных распараллелим стандартный алгоритм сортировки слиянием и построим схемы обоих алгоритмов.

2.1 Параллельная сортировка слиянием

Легко заметить, что массив при каждом рекурсивном вызове делится на две примерно равные части и производится сортировка сначала левой, а затем правой частей массива. Это, значит что последовательные действия можно заменить на параллельные. При каждом рекурсивном вызове будет создаваться поток, который займется сортировкой левой части исходного массива в то время, как вызвавший его поток будет сортировать правую часть. Таким образом можно добиться максимального количества параллельно работающих потоков. При такой реализации каждый поток будет работать только с вверенной ему частью исходного массива, то есть не возникнет проблем, связанных с множественным доступом к данным.

Поскольку на реальной вычислительной машине невозможно добиться создания бесконечного количества потоков, нужно отслеживать текущий уровень иерархии рекурсивных вызовов для своевременного прекращения создания потоков. При достижении максимально возможного количества созданных потоков каждый из них будет параллельно выполнять стандартный алгоритм сортировки слиянием.

2.2 Схемы алгоритмов

На рисунках 2.1–2.4 представлены схемы алгоритмов сортировки слиянием.

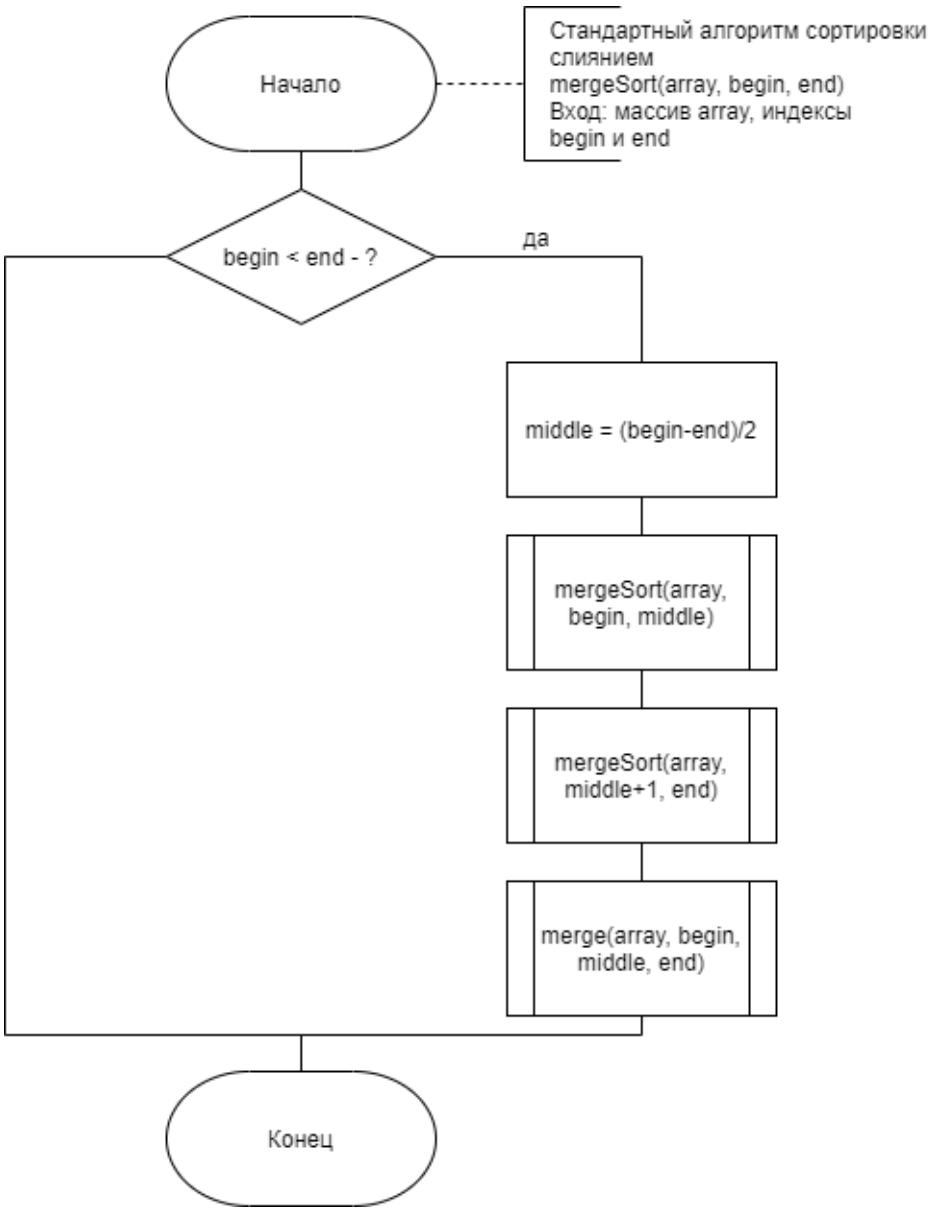


Рисунок 2.1 — Стандартный алгоритм сортировки слиянием

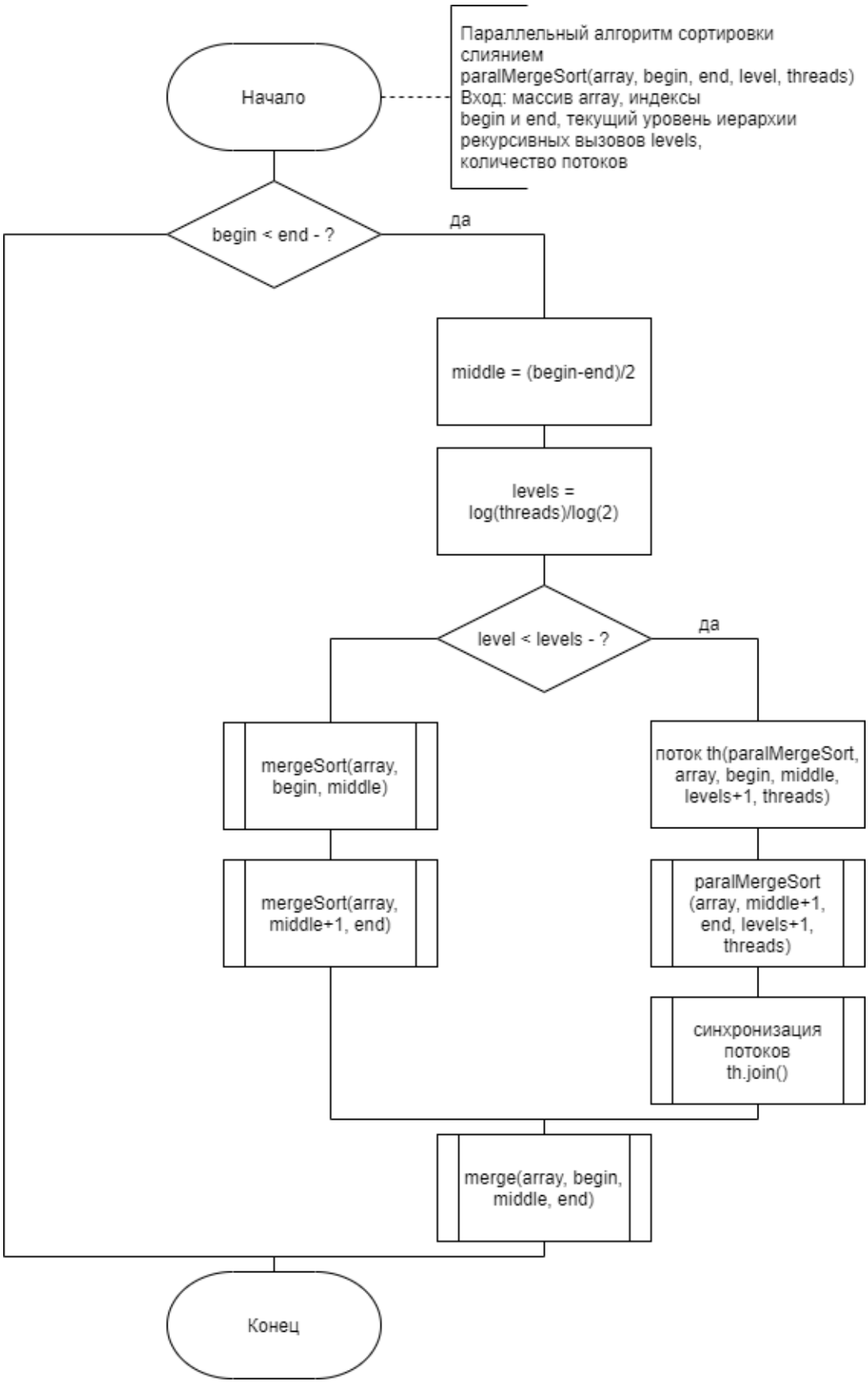


Рисунок 2.2 — Параллельная сортировка слиянием

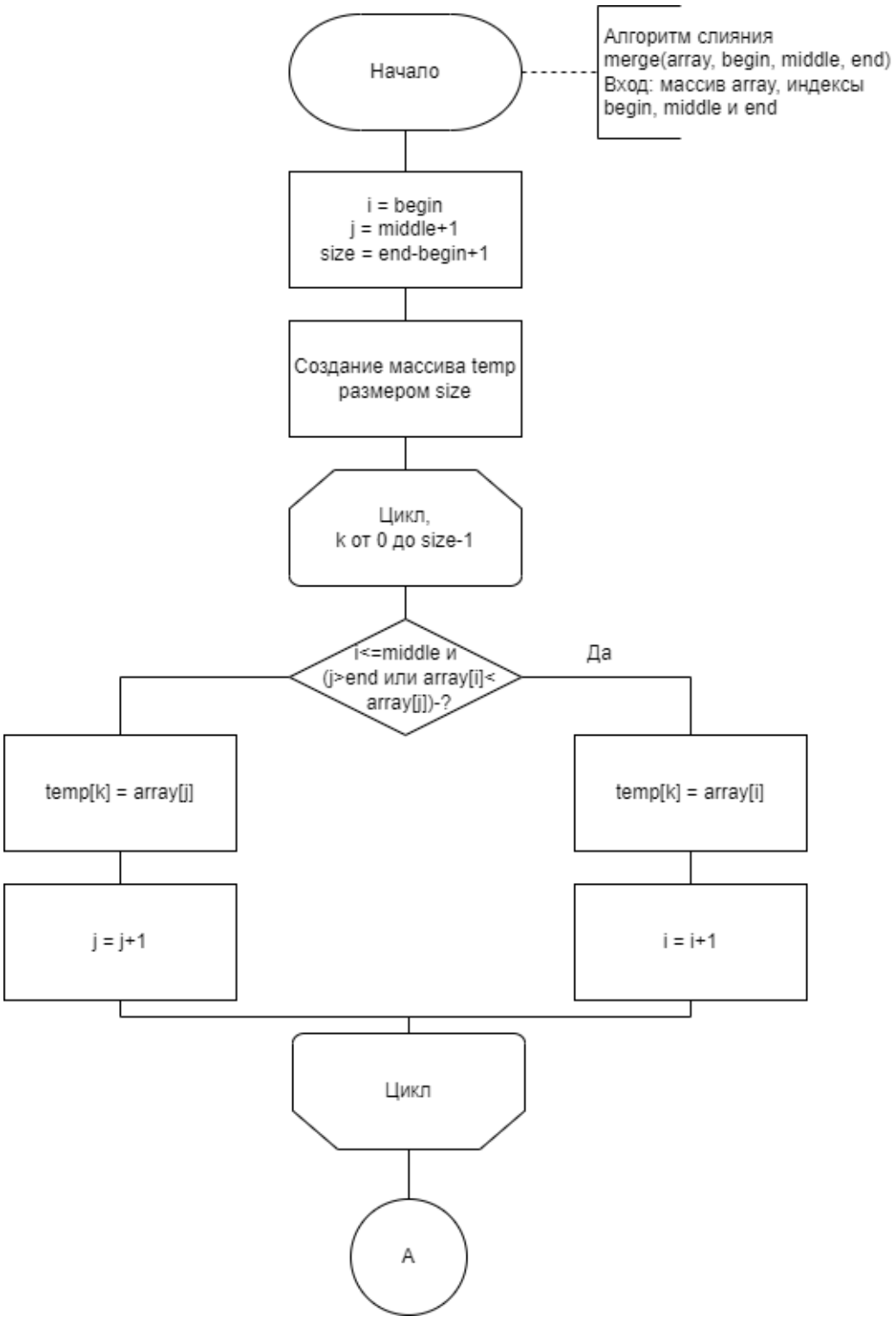


Рисунок 2.3 — Алгоритм слияния. Часть 1

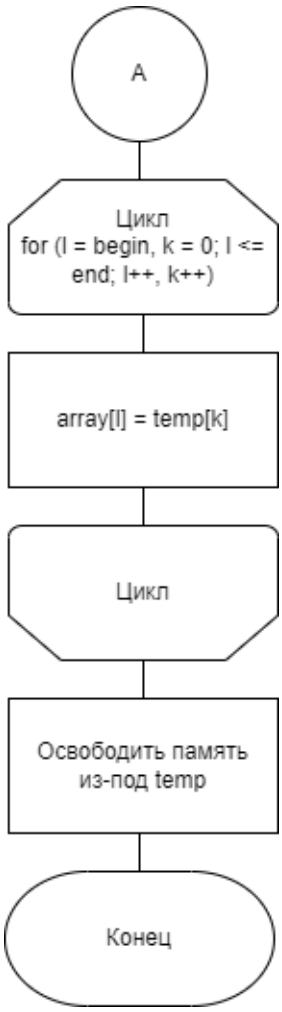


Рисунок 2.4 — Алгоритм слияния. Часть 2

2.3 Типы и структуры данных

Для хранения исходного набора значений используется массив целых чисел. Запись результата будет производиться в исходный массив. Каждый поток будет работать только со своей частью исходного массива, таким образом избегается возможность критических ситуаций и потери информации.

2.4 Способ тестирования

Реализуемое программное обеспечение будет протестировано методом `черного ящика`. Для тестирования были выделены следующие классы эквивалентности:

1. Неверный выбор режима работы программы — пустой ввод, нецифровой символ.

2. Верный выбор режима работы программы — цифра из диапазона [1..3] или число, выходящее за пределы указанного диапазона.
3. Неверный выбор количества потоков при параллельной сортировке — пустой ввод, нецифровой символ или число, не входящее в набор указанных доступных значений.
4. Верный выбор количества потоков при параллельной сортировке — число из набора $(2, 4, 8, 16)$.
5. Неверный ввод размера исходного массива — пустой ввод, нецифровой символ или целое неположительное или вещественное число.
6. Верный ввод размера исходного массива — целое положительное число.
7. Неверный ввод элемента исходного массива — пустой ввод, нецифровой символ или вещественное число.
8. Верный ввод элемента исходного массива — целое число.

2.5 Тестовые данные

В таблицах 2.1–2.4 представлены тестовые данные.

Таблица 2.1 — Функциональные тесты выбора режима работы программы

Номер тестового случая	Ввод	Ожидаемый результат
1	_[1]	Неверный ввод
2	α	Неверный ввод
3	1	Введите размер массива
4	2	Введите количество потоков
5	3	Таблица замеров времени

Таблица 2.2 — Функциональные тесты выбора количества потоков

Номер тестового случая	Ввод	Ожидаемый результат
1	_[1]	Неверный ввод
2	α	Неверный ввод
3	1	Неверный ввод
4	2	Введите размер массива

[1]Пустой ввод

Таблица 2.3 — Функциональные тесты ввода размера исходного массива

Номер тестового случая	Ввод	Ожидаемый результат
1	_[1]	Неверный ввод
2	α	Неверный ввод
3	0	Неверный ввод
4	2.2	Неверный ввод
5	2	Введите элементы массива

Таблица 2.4 — Функциональные тесты ввода элементов исходного массива

Номер тестового случая	Ввод	Ожидаемый результат
1	_[1]	Неверный ввод
2	α	Неверный ввод
3	2.2	Неверный ввод
4	0	...[2]

2.6 Структура программного обеспечения

Программное обеспечение разработано с использованием структурного подхода, поскольку использование объектов в данном случае не дает преимуществ. Программа содержит функцию `mergeSort`, принимающая на вход указатель на исходный массив, индекс начала и индекс конца обрабатываемой части массива. Она реализует стандартный алгоритм сортировки слиянием. Функция `merge` реализует слияние двух отсортированных частей массива в один отсортированный. На вход она принимает указатель на исходный массив, индекс начала и конца сливаемой части массива, а также индекс конца первого подмассива. Функция `paralMergeSort` принимает те же параметры, что и функция `mergeSort`, а также максимальное количество создаваемых потоков и текущий уровень иерархии рекурсивных вызовов. Все три функции ничего не возвращают, они лишь изменяют вверенные им части исходного массива.

[1]Пустой ввод

[2]Ожидание ввода следующего элемента

2.7 Вывод по конструкторской части

Были разработаны схемы стандартного алгоритма сортировки слиянием и его распараллеленного аналога, а также подготовлены данные для тестирования программного обеспечения.

3 Технологическая часть

В данном разделе приведены средства реализации и листинги кода.

3.1 Требования к ПО

К программе предъявляется ряд требований:

- требования ко вводу
  - должен быть выбран режим работы программы (целое число в дапазоне [1..3]);
  - должны быть введены следующие значения: количество потоков при необходимости (целое число из набора $(2, 4, 8, 16)$), размер массива (целое положительное число), элементы массива (целые числа).
- требования к выводу
  - результат сортировки;
  - таблица времен работы реализаций алгоритмов на случайных значениях при разных размерах массивов и разных количествах потоков в миллисекундах.
- ограничения работы программы
  - при вводе режима работы программы, выходящего за пределы указанного диапазона, программа завершает работу.
- функциональные требования к программному обеспечиванию:
  - при запуске программа должна выводить меню с возможными режимами работы;
  - программа должна сортировать входной массив целых чисел стандартным алгоритмом сортировки слиянием или его распараллеленным аналогом;

– в исследовательском режиме программа должна замерять время выполнения реализаций стандартного алгоритма сортировки слиянием и его распараллеленного аналога при различных размерах массивов и с разным количеством используемых потоков в последнем случае.

3.2 Средства реализации

Для реализации ПО был выбран язык программирования `C++` [1]. Это обусловлено наличием широкого спектра возможностей для работы с потоками и распараллеливания алгоритмов.

В качестве среды разработки была выбрана `Visual Studio Code` [3]. Достаточный опыт работы в этой среде, удобства написания кода и его автодополнения стали ключевыми при выборе.

3.3 Реализация алгоритмов

В листингах 3.1 - 3.3 приведена реализация рассматриваемых алгоритмов.

Листинг 3.1 — Стандартный алгоритм сортировки слиянием

```
void mergeSort(int *array, int begin, int end) {
    if (begin < end) {
        int middle = (end + begin) / 2;
        mergeSort(array, begin, middle);
        mergeSort(array, middle+1, end);
        merge(array, begin, middle, end);
    }
}
```

Листинг 3.2 — Параллельная сортировка слиянием

```
void paralMergeSort(int *array, int begin, int end, int
  level, int threads) {
    if (begin < end) {
        int middle = (end + begin) / 2;
        int levels = (int)(log(((double)threads) / log
          (2.0)));
        if (level < levels) {
            thread th(paralMergeSort, array, begin,
              middle, level+1, threads);
            paralMergeSort(array, middle+1, end, level+1,
               threads);
            th.join();
        } else {
            mergeSort(array, begin, middle);
            mergeSort(array, middle+1, end);
        }
        merge(array, begin, middle, end);
    }
}
```

Листинг 3.3 — Функция слияния двух отсортированных массивов

```
void merge(int *array, int begin, int middle, int end) {
    int i = begin, j = middle + 1, size = end - begin + 1;
    int *temp = (int *)malloc(sizeof(int)*size);
    for (int k = 0; k < size; k++) {
        if (i <= middle &&
            (j > end || array[i] < array[j])) {
            temp[k]=array[i];
            i++;
        } else {
            temp[k]=array[j];
            j++;
        }
    }
    for (int l = begin, k = 0; l <= end; l++, k++) {
        array[l] = temp[k];
    }
    free(temp);
}
```

3.4 Вывод по технологической части

В данном разделе были реализованы стандартный алгоритм сортировки слиянием и его распараллеленный аналог.

4 Исследовательская часть

В этом разделе будет исследовано быстродействие разработанных реализаций алгоритмов.

4.1 Технические характеристики

Ниже приведены технические характеристики устройства, на котором было проведено тестирование ПО:

- операционная система Windows 10 64-разрядная;
- оперативная память 16 ГБ;
- процессор Intel(R) Core(TM) i5-4690 @ 3.50ГГц.

4.2 Время выполнения реализаций алгоритмов

Чтобы точно оценить время выполнения реализаций алгоритмов была использована функция `std::chrono::high_resolution_clock::now()` [2], которая возвращает точку, указывающую на текущее время. Так, с помощью двух последовательных вызовов можно определить, в течение какого времени выполнялась функция.

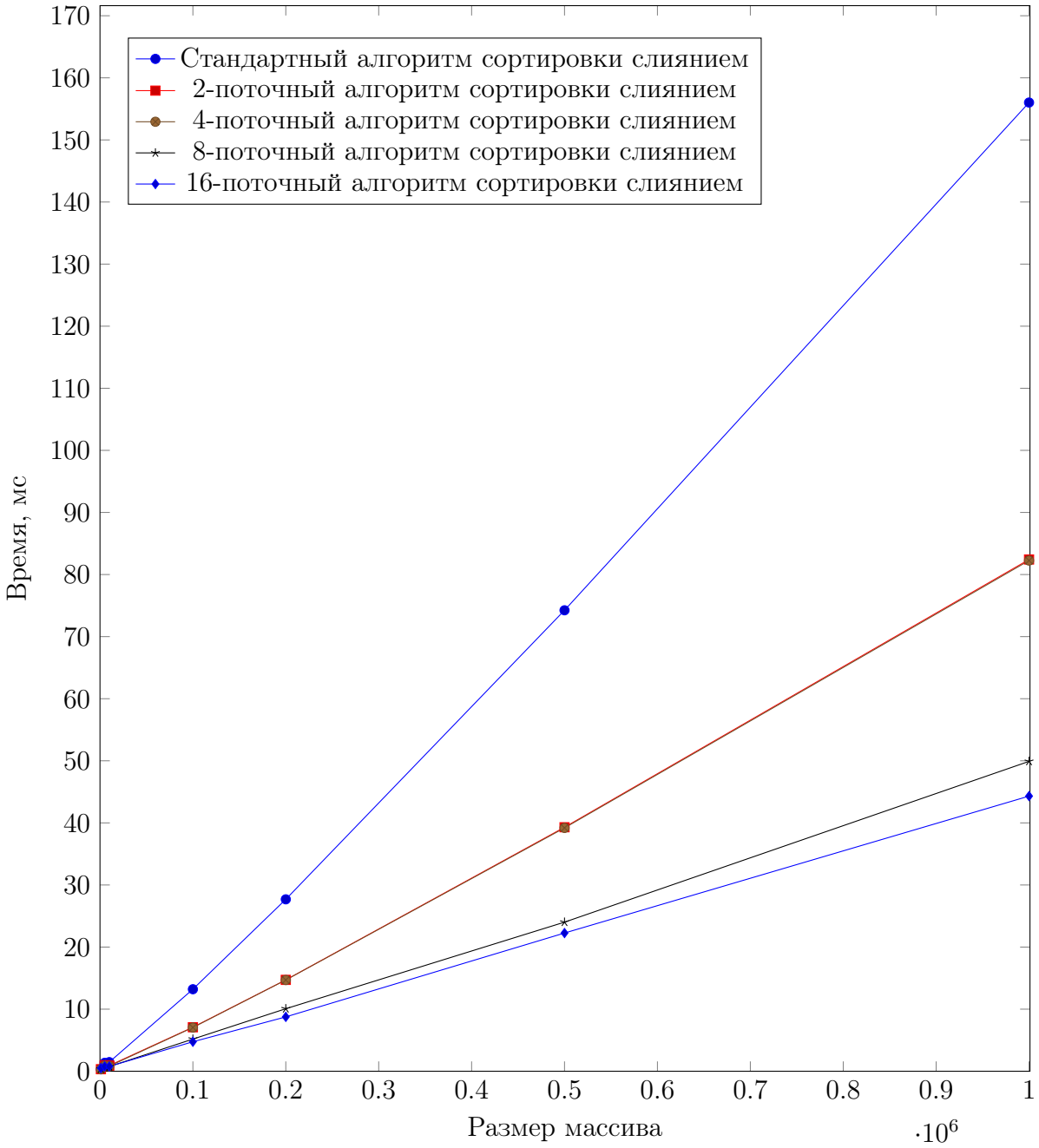
В таблице 4.1 и на графике 4.1 показаны результаты замеров.

Таблица 4.1 — Зависимость времени выполнения реализаций алгоритмов сортировки слиянием от размера массива (в миллисекундах)

Размер массива	Стандарт[1]	2 потока[2]	4 потока[3]	8 потоков[4]	16 потоков[5]
1000	0.310042	0.328200	0.327511	0.361685	0.492158
5000	1.350680	0.964878	0.874131	0.761405	0.778045
10000	1.468820	0.914586	0.820653	0.745670	0.777463
100000	13.208900	7.088850	7.042640	5.179420	4.749100
200000	27.689400	14.715900	14.693300	10.069600	8.754370
500000	74.242100	39.297600	39.188600	24.013900	22.259800
1000000	156.021000	82.423000	82.253000	49.909500	44.313600

[1]Стандартный алгоритм сортировки слиянием
[2]Параллельная сортировка слиянием с 2 потоками
[3]Параллельная сортировка слиянием с 4 потоками
[4]Параллельная сортировка слиянием с 8 потоками
[5]Параллельная сортировка слиянием с 16 потоками

График 4.1 — Время выполнения реализаций алгоритмов сортировки слиянием



4.3 Вывод по исследовательской части

По результатам замеров можно сказать, что на массивах больших размеров многопоточный алгоритм в несколько раз быстрее, чем стандартный. Но на малых размерах (до 1000 элементов) стандартный работает быстрее, поскольку задача для каждого потока оказывается настолько простой, что намного больше времени уходит на создание самих потоков.

Заключение

В ходе проделанной работы была достигнута поставленная цель и решены следующие задачи:

- выбран и изучен стандартный алгоритм, работающий с памятью;
- разработаны схемы стандартного алгоритма и его распараллеленной версии;
- реализованы эти алгоритмы;
- проведено тестирование реализаций алгоритмов;
- проведен сравнительный анализ скорости работы реализаций алгоритмов с учетом возможности выделения разного количества потоков.

Было доказано, что имеет смысл выделять потоки только в том случае, если каждому потоку будет дана сложная задача, в противном случае намного больше времени будет затрачиваться на создание потоков.

Литература

[1] C++ [Электронный ресурс]. Режим доступа: https://isocpp.org/. Дата обращения: 11.11.2021.

[2] Функция now() [Электронный ресурс]. Режим доступа: https://en.cppreference.com/w/cpp/chrono/high-resolution-clock/now. Дата обращения: 11.11.2021.

[3] Visual Studio Code - Code Editing [Электронный ресурс]. Режим доступа: https://code.visualstudio.com. Дата обращения: 11.11.2021.

[4] Сортировки [Электронный ресурс]. Режим доступа: https://function-x.ru/cpp-algoritmy-sortirovki.html. Дата обращения: 11.11.2021.