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 дисциплине "Анализ алгоритмов"

Тема Параллельное программирование

Студент Егорова П.А.

Группа ИУ7-54Б

Преподаватели Волкова Л.Л., Строганов Ю.В., Строганов Д.В.

Москва — 2022 г.

Оглавление

Введение

Многопоточность — способность центрального процессора (CPU) или одного ядра в многоядерном процессоре одновременно выполнять несколько процессов или потоков, соответствующим образом поддерживаемых операционной системой.

Многопоточность (как доктрину программирования) не следует путать ни с многозадачностью, ни с многопроцессорностью, несмотря на то, что операционные системы, реализующие многозадачность, как правило, реализуют и многопоточность.

Достоинства:

- облегчение программы посредством использования общего адресного пространства;
- меньшие затраты на создание потока в сравнении с процессами;
- повышение производительности процесса за счёт распараллеливания процессорных вычислений;
- если поток часто теряет кэш, другие потоки могут продолжать использовать неиспользованные вычислительные ресурсы.

Недостатки:

- несколько потоков могут вмешиваться друг в друга при совместном использовании аппаратных ресурсов [1];
- с программной точки зрения аппаратная поддержка многопоточности более трудоемка для программного обеспечения [2];
- проблема планирования потоков.

Однако несмотря на количество недостатков, перечисленных выше, многопоточная парадигма имеет большой потенциал на сегодняшний день и при должном написании кода позволяет значительно ускорить однопоточные алгоритмы.

В лабораторной работе №3 был проведен сравнительный анализ трудоемкости трех видов сортировки: блочной, перемешиванием и бинарным деревом. Было выявлено, что блочная сортировка является независимой от входных данных: практически одинаковое время при работе с массивом чисел, отсортированным по невозрастанию, неубыванию, и массиве случайных чисел. Также сортировка зачастую обгоняла конкурентные. Но возможно ли сократить и без того небольшое время работы данной сортировки?

Целью данной лабораторной работы является получение навыков организации параллельных вычислений на базе потоков.

Чтобы достичь намеченной цели необходимо решить следующие задачи:

- изучить понятие параллельных вычислений;
- изучить виды параллелизма;
- реализовать последовательный и параллельный алгоритм блочной сортировки;
- сравнить временные характеристики реализованных алгоритмов экспериментально на разных размерах входных данных;
- сравнить временные характеристики реализованных алгоритмов экспериментально при разном количестве рабочих потоков.

1 Аналитическая часть

1.1 Параллелизм

Параллелизм предполагает выполнения нескольких задач в момент времени. Покажем стрелкой зависимость данных, используемых в задаче Б, от данных, используемых в задаче А, как показано на рисунке 1.1. То есть данные, рассчитанные в результате выполнения задачи А, используются при выполнении задачи Б.

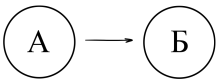

Рис. 1.1: Зависимость задачи А от задачи Б по данным

Выделяют два вида параллелизма: конечный, когда можно параллельно выполнять отдельные инструменты или их набор, и массовый, предполагающий параллельное исполнение итераций циклов. В свою очередь массовый параллелизм делится еще на два: покоординатный – вычисления в пределах одной координаты не зависят по данным от вычислений для другого значения координаты, и скошенный. Зависимость по данным задач представлена на рисунке 1.2.

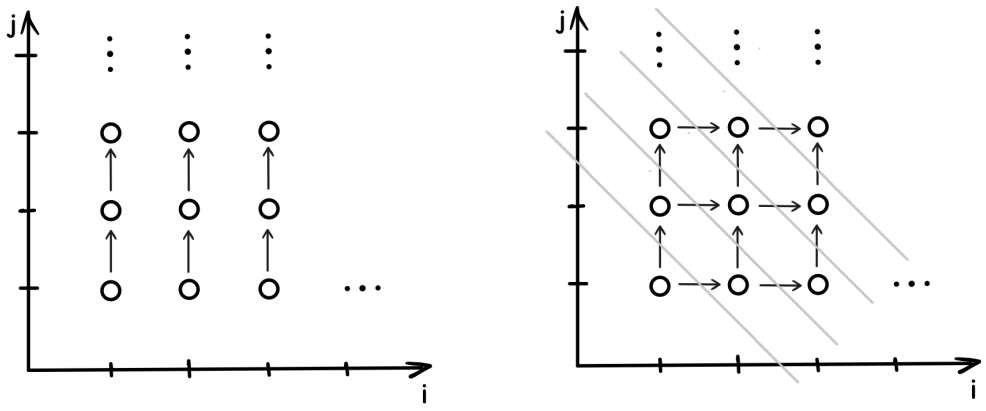

Рис. 1.2: Покоординатный и скошенный параллелизм

1.2 Блочная сортировка

Блочная сортировка [3] – это алгоритм сортировки, который разделяет несортированные элементы массива на несколько групп, называемых блоками или корзинами. Затем каждая корзина сортируется с использованием любого из подходящих алгоритмов сортировки или рекурсивного применения того же алгоритма блочной сортировки.

Мной были найдены реализации данной сортировки только для положительных чисел, поэтому этап поиска интервала – размера блока был усовершенствован, дабы алгоритм был применим и для отрицательных чисел.

В итоге отсортированные сегменты объединяются для формирования окончательного отсортированного массива.

Исходя из общей концепции, алгоритм может быть разбит на следующие части:

- определение числового интервала – объема блока;
- распределение чисел массива по корзинам;
- сортировка каждого отдельного блока;

Последний пункт является циклом, на каждой итерации которого рассматривается отдельная корзина: то есть данные и операции над ними на одной итерации цикла не влияют на данные на другой. Таким образом этот этап алгоритма сортировки может быть реализован с помощью распараллеливания – покоординатный вид массового параллелизма.

1.3 Вывод

В данном разделе были рассмотрены основополагающие материалы, которые в дальнейшем потребуются при параллельной и однопоточной реализации алгоритма блочной сортировки.

2 Конструкторская часть

В данном разделе будет рассмотрены схемы исследуемого алгоритма.

2.1 Схемы алгоритмов

На рисунке 2.1 представлена схема однопоточного алгоритма блочной сортировки.

На рисунке 2.2 представлена схема алгоритма сортировки вставками, используемой для сортировки каждого блока.

На рисунке 2.3 представлена схема распараллеленного алгоритма блочной сортировки.

2.2 Вывод

На основе теоретических данных, полученных из аналитического раздела, была построена схема стандартного алгоритма блочной сортировки, а также, после разделения алгоритма на этапы, была предложена схема параллельного выполнения этапов.

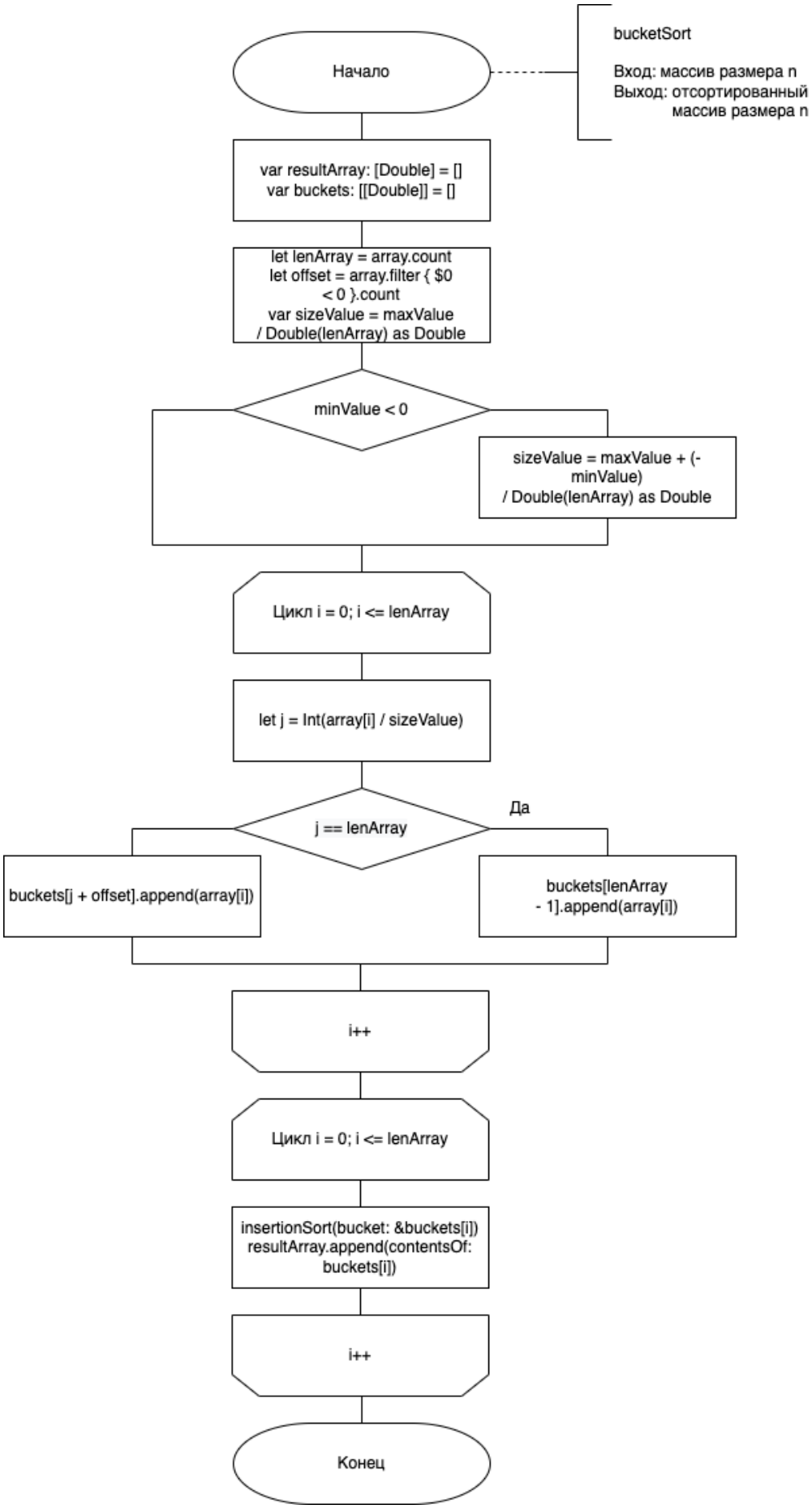

Рис. 2.1: Схема алгоритма блочной сортировки

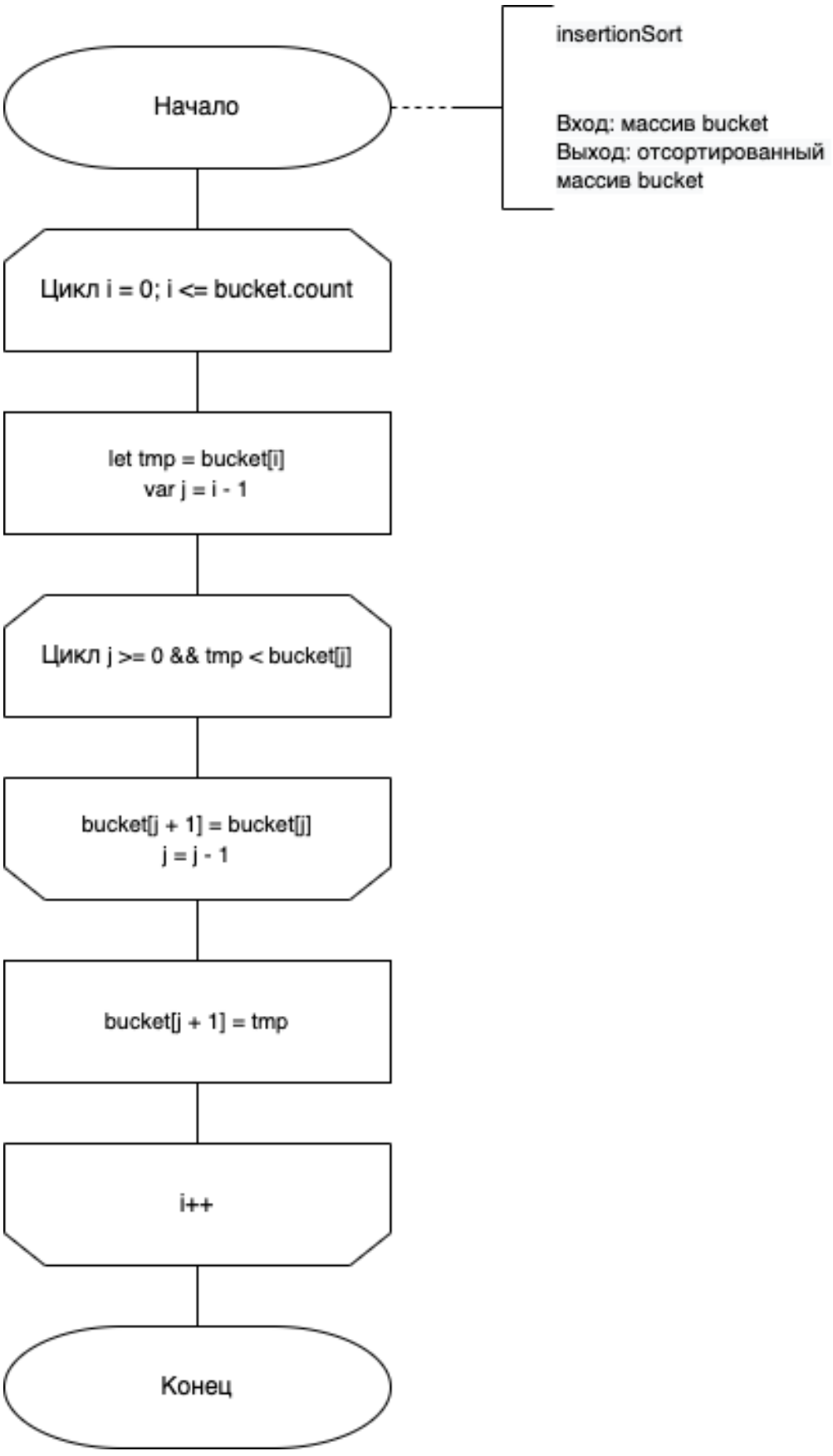

Рис. 2.2: Схема алгоритма блочной сортировки

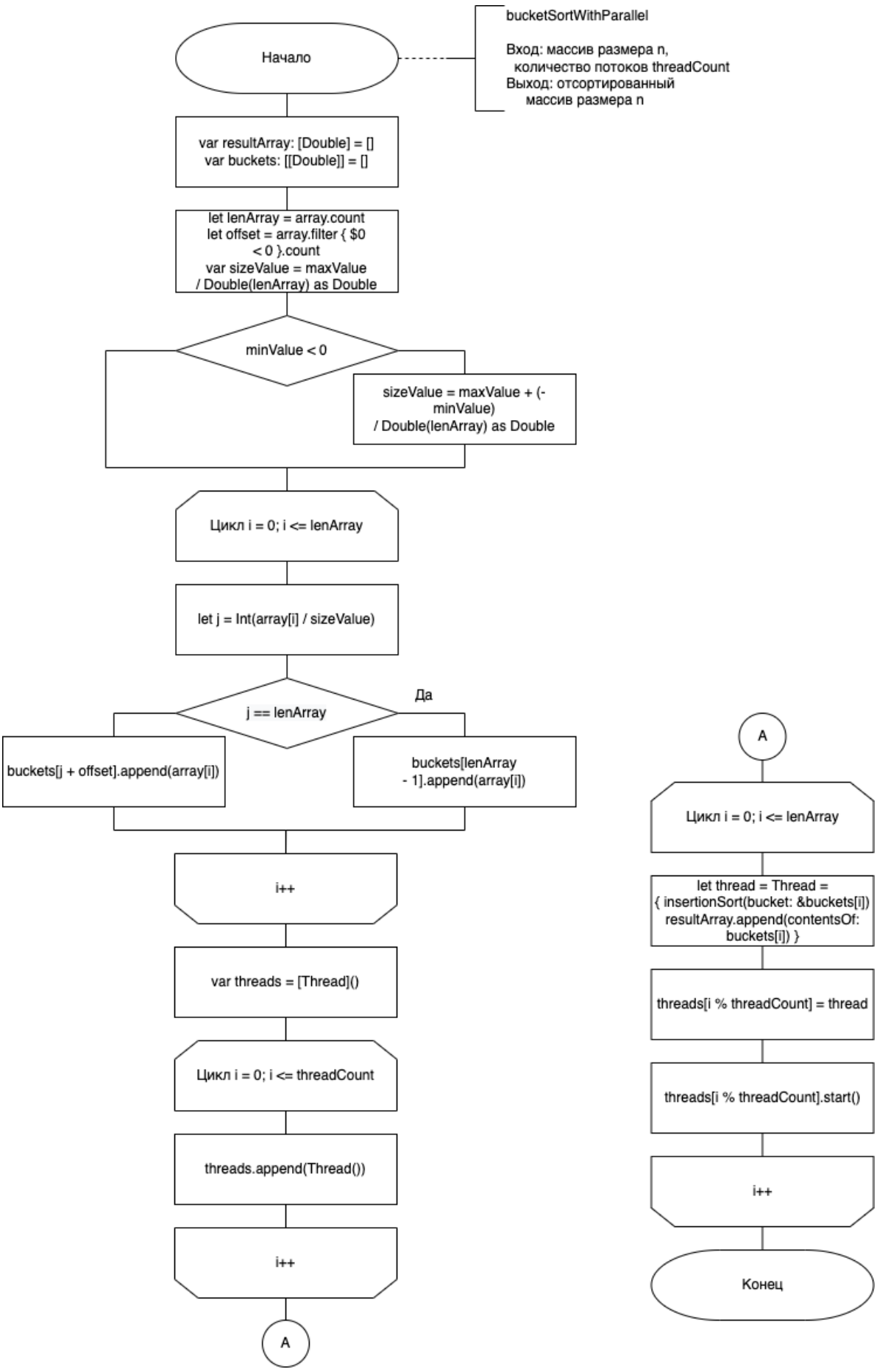

Рис. 2.3: Схема алгоритма блочной сортировки

3 Технологическая часть

В данном разделе производится выбор средств реализации, а также приводятся требования к программному обеспечению (ПО), листинги реализованных алгоритмов.

3.1 Требования к ПО

На вход программе подается массив сравниваемых элементов, на выходе должен быть получен отсортированный массив. Результирующий массив вычислен с помощью однопоточного алгоритма блочной сортировки и алгоритма, использующего параллелизм. Также необходимо вывести затраченное каждым алгоритмом процессорное время с учетом количества потоков второй реализации.

3.2 Выбор средств реализации

В качестве языка программирования для реализации данной лабораторной работы был выбран язык Swift [4]. Данный язык создан компаний Apple для разработки программного обеспечения для macOS, iOS, watchOS и tvOS. Выбор обоснован желанием расширить знания в области применения данного языка, а также возможностью создать на основе написанного кода приложение.

Кроме того, в Swift есть фреймворк CoreFoundation [5], который предоставляет функции для замера процессорного времени. В качестве среды разработки выбран XCode [6].

Для работы с потоками будет использоваться класс NSThread [7] библиотеки Foundation.

3.3 Листинги кода

В листингах 3.1 и 3.3 представлены реализации рассматриваемых алгоритмов блочной сортировки: однопоточная и многопоточная.

В Листинге 3.1 показана реализация однопоточного алгоритма блочной сортировки.

```
func bucketSort(_ array: [Double]) -> [Double]  {
    var resultArray: [Double] = []
    let maxValue = array.max()!
    let minValue = array.min()!
    let lenArray = array.count
    let offset = array.filter { $0 < 0 }.count
    var sizeValue = maxValue / Double(lenArray) as Double
    if minValue < 0 { sizeValue = maxValue + (-minValue) / Double(lenArray
   ) as Double }
    var buckets: [[Double]] = []
    for _ in 0..<lenArray { buckets.append([]) }
    for i in 0..<lenArray {
        let j = Int(array[i] / sizeValue)
        if j != lenArray { buckets[j + offset].append(array[i]) }
        else { buckets[lenArray - 1].append(array[i]) }
    }
    for i in 0..<lenArray {
        insertionSort(bucket: &buckets[i])
        resultArray.append(contentsOf: buckets[i])
    }
    return resultArray
}
```

Листинг 3.1: Функция алгоритма блочной сортировки однопоточная

В Листинге 3.2 показана реализация алгоритма сортировки вставками.

```
func insertionSort(bucket: inout [Double]) {
    for i in 0..<bucket.count {
        let tmp = bucket[i]
        var j = i - 1
        while (j >= 0 && tmp < bucket[j]) {
            bucket[j + 1] = bucket[j]
            j = j - 1
        }
        bucket[j + 1] = tmp }
}
```

Листинг 3.2: Функция алгоритма сортировки вставками

В Листинге 3.3 показана реализация многопоточного алгоритма блочной сортировки.

```
func bucketSort(_ array: [Double]) -> [Double]  {
    var resultArray: [Double] = []
    let maxValue = array.max()!
    let minValue = array.min()!
    let lenArray = array.count
    let offset = array.filter { $0 < 0 }.count
    var sizeValue = maxValue / Double(lenArray) as Double
    if minValue < 0 { sizeValue = maxValue + (-minValue) / Double(lenArray
   ) as Double }
    var buckets: [[Double]] = []
    for _ in 0..<lenArray { buckets.append([]) }
    for i in 0..<lenArray {
        let j = Int(array[i] / sizeValue)
        if j != lenArray {
            buckets[j + offset].append(array[i])
        } else { buckets[lenArray - 1].append(array[i]) }
    }
    var threads = [Thread]()
    for _ in 0..<threadCount { threads.append(Thread()) }

    for i in 0..<lenArray {
        let thread = Thread {
            insertionSort(bucket: &buckets[i])
            resultArray.append(contentsOf: buckets[i])
        }
        let j = i % threadCount
        threads[j] = thread
        threads[j].start()
    }
    return resultArray
}
```

Листинг 3.3: Функция алгоритма блочной сортировки многопоточная

В листинге 3.2 представлена реализации алгоритма сортировки вставками, применяемая в каждому выделенному блоку на последнем этапе блочной сортировки.

3.4 Функциональные тесты

В таблице 3.1 приведены тесты для функций, реализующих алгоритмы сортировки. Все тесты пройдены успешно.

Входной массив	Результат	Ожидаемый результат
$[15, 25, 35, 45]$	$[15, 25, 35, 45]$	$[15, 25, 35, 45]$
$[55, 45, 35, 25]$	$[25, 35, 45, 55]$	$[25, 35, 45, 55]$
$[-10, -20, -30, -25]$	$[-30, -25, -20, -10]$	$[-30, -25, -20, -10]$
$[40, -10, -30, 75]$	$[-30, -10, 40, 75]$	$[-30, -10, 40, 75]$
$[100]$	$[100]$	$[100]$
$[-20]$	$[-20]$	$[-20]$
$[1.1, 2.2, 3.3, 4.4]$	$[1.1, 2.2, 3.3, 4.4]$	$[1.1, 2.2, 3.3, 4.4]$
$[1.1, -2.2, 3.3, -4.4]$	$[-4.4, -2.2, 1.1, 3.3]$	$[-4.4, -2.2, 1.1, 3.3]$
$[-1.1, -2.2, -3.3, -4.4]$	$[-4.4, 3.3, -2.2, -1.1]$	$[-4.4, -3.3, -2.2, -1.1]$
$[10, 10]$	$[10, 10]$	$[10, 10]$

Таблица 3.1: Тестирование функций

3.5 Вывод

В данном разделе были разработаны исходные коды алгоритмов: однопоточная и многопоточная реализации алгоритма блочной сортировки, а также вспомогательный алгоритм – сортировка вставками.

4 Исследовательская часть

Технические характеристики устройства, на котором было произведено тестирование разработанного ПО:

- операционная система: macOS Big Sur 11.6.4;
- оперативная память: 8 Гб;
- 2,4 Ггц 2-ядерный процессор Intel Core i5 [9].

Во время тестирования ноутбук был включен в сеть питания.

4.1 Время выполнения реализаций алгоритмов

В таблице 4.1 приведено время (мс) работы реализаций параллельного алгоритма блочной сортировки для массивов случайных чисел размеров от 10000 до 50000 элементов.

Таблица 4.1: Результаты замеров времени многопоточного алгоритма блочной сортировки

Количество потоков	10000	20000	30000	40000	50000
1	1.3204	6.7280	18.5843	32.6369	51.2648
4	1.0098	2.8004	5.5141	9.4126	14.7643
8	1.3321	3.9422	7.2482	9.7218	15.2290
16	1.2935	4.0348	8.3376	11.5191	15.9671
32	1.3028	4.0505	8.3464	13.4814	17.0147
64	1.2980	4.0466	8.4265	13.9359	19.9160

Наилучшее время параллельные алгоритмы показали на 4 потоках, что соответствует количеству логических ядер компьютера, на котором проводилось тестирование. Время работы (мс) алгоритма однопоточной реализации и реализации с использованием оптимального количества ядер представлены в таблице 4.2.

Таблица 4.2: Результаты замеров времени однопоточной и многопоточной реализаций алгоритма сортировки

Размер	1 поток	4 потока
10000	1.3653	1.0098
20000	6.7280	2.8004
30000	18.5843	5.5141
40000	32.6369	9.4126
50000	51.2648	14.7643

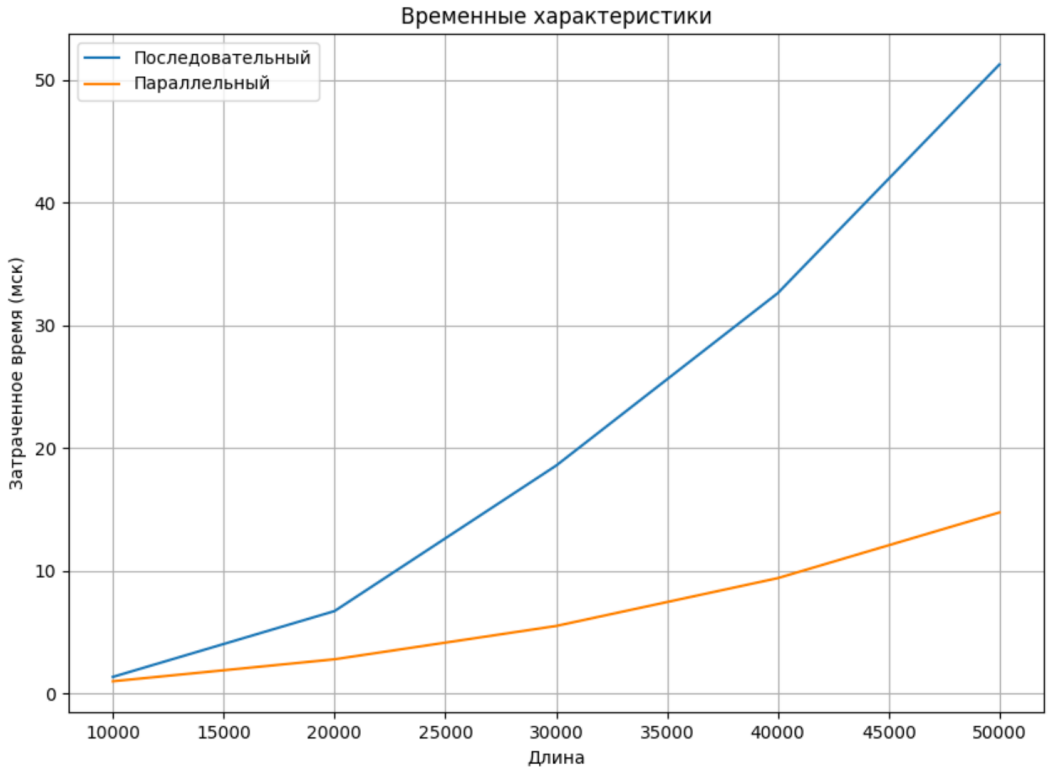

Рис. 4.1: Время работы сортировок

На рисунке 4.1 изображены графики времени работы однопоточного алгоритма сортировки и реализации с 4 потоками.

На рисунке 4.2 изображены графики зависимости времени работы алгоритмов сортировок от количества элементов массивов.

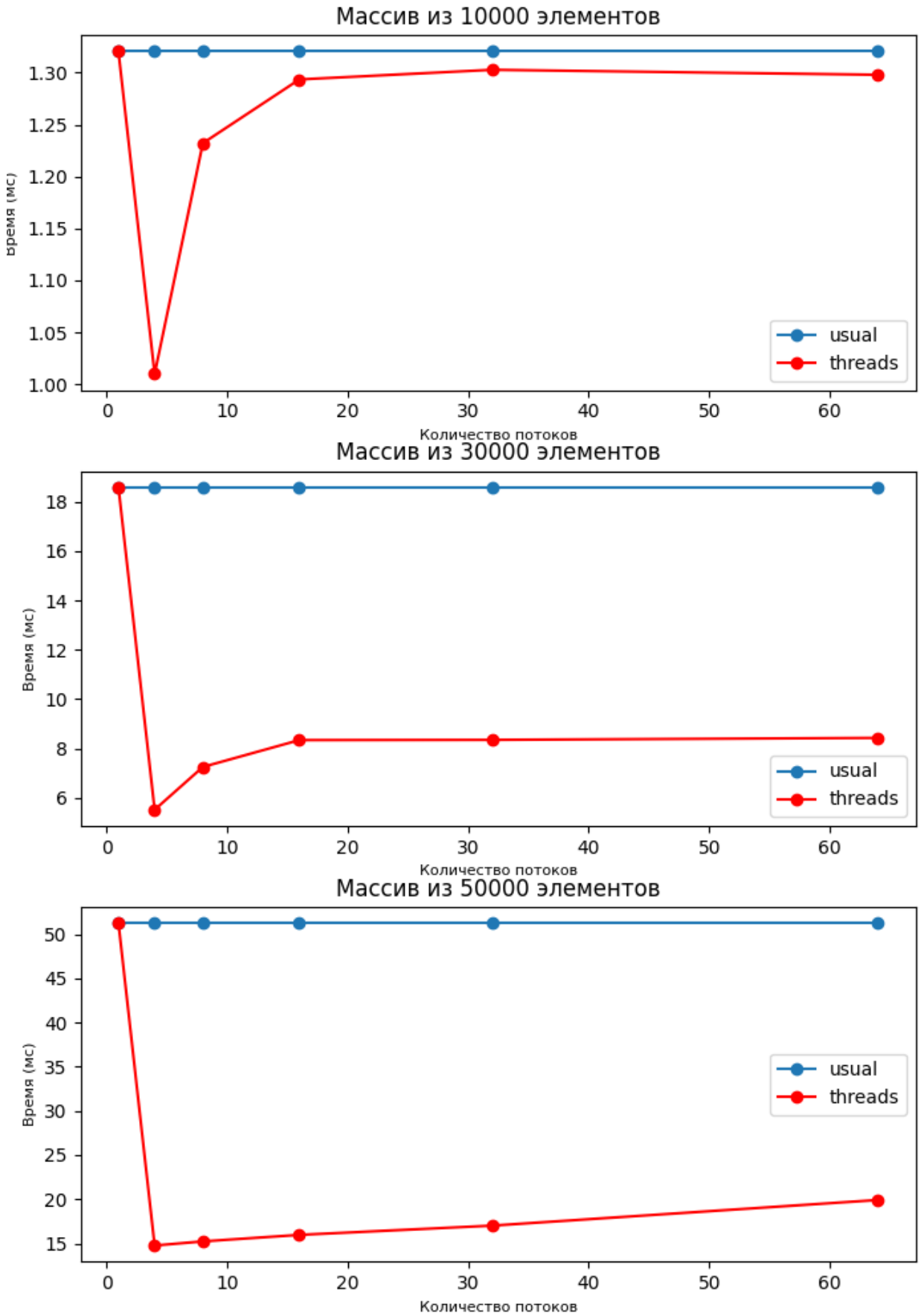

Рис. 4.2: Время работы сортировок

Вывод

В данном разделе было произведено сравнение алгоритмов блочной сортировки при однопоточной реализации и многопоточной.

Экспериментально получено, что при реализации алгоритма блочной сортировки с использованием параллелизма эффективнее по времени, чем конкурентная однопоточная. Тестирование показало, что распараллеливание алгоритма помогает ускорить работу, а самым выгодным является использование 4 потоков, что совпадает с количеством логических ядер машины, на которой производилось тестирование. В среднем многопоточная реализация выигрывает у однопоточной в 3.5 раза.

Заключение

В ходе выполнения лабораторной работы были решены следующие задачи:

- рассмотрены основы параллельных вычислений;
- полученные знания применены для однопоточной и многопоточной реализации алгоритма блочной сортировки;
- определен вид параллелизма в данной реализации – покоординатный вид массового параллелизма;
- произведен сравнительный анализ однопоточной и параллельной реализации алгоритма сортировки при разном количестве потоков;
- произведен сравнительный анализ однопоточной и параллельной (при оптимальном количестве потоков) реализации алгоритма сортировки на разных размерах массива, поступающего на вход.

Литература

[1] Mario Nemirovsky D. M. T. Multithreading Architecture // Morgan and Claypool Publishers. 2013.

[2] Olukotun K. Chip Multiprocessor Architetture – Techniques to Improve Throughput and Latency // Morgan and Claypool Publishers. 2007. p. 154.

[3] Блочная сортировка [Электронный ресурс]. Режим доступа: https://www.programiz.com/dsa/bucket-sort

[4] Swift: Swift | Мультипарадигмальный компилируемый язык [Электронный ресурс]. Режим доступа: https://www.apple.com/swift/

[5] Core Foundation: Core Foundation | Time Utilities [Электронный ресурс]. Режим доступа: https://developer.apple.com/documentation/corefoundation/time_utilities

[6] Xcode: Xcode [Электронный ресурс]. Режим доступа: https://developer.apple.com/xcode/

[7] NSThread | Apple Developer Documentation [Электронный ресурс]. Режим доступа: https://developer.apple.com/documentation/foundation/nsthread

[8] AppCode: AppCode | Smart Swift/Objective-C IDE for iOS [Электронный ресурс]. Режим доступа: https://www.jetbrains.com/objc/

[9] Intel Core i5 3M Cache 2.4 GHz Спецификации продукции [Электронный ресурс]. Режим доступа: https://ark.intel.com/content/www/ru/ru/ark/products/47341/intel-core-i5520m-processor-3m-cache-2-40-ghz.html