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 №5
по дисциплине «Анализ алгоритмов»

Тема: Конвейерные вычисления
Студент: Карпова Е. О.
Группа: ИУ7-52Б
Оценка (баллы): _____
Преподаватели: Волкова Л. Л., Строганов Ю. В.

Москва — 2022 г.

Оглавление

Введение

В процессе реализации ПО зачастую появляется необходимость обработки одного набора данных в несколько этапов. Если эти этапы связаны между собой по данным, то есть данные, полученные на предыдущем этапе, являются входными для следующего, то увеличить эффективность поможет использование конвейерная обработка данных [7].

В работе рассмотрены два типа конвейерной обработки данных: линейная и параллельная. При линейной конвейеризации одновременно в системе с конвейером может находиться только одна заявка: пока её обработка не завершена, другие заявки не могут попасть в конвейер. Тогда частым явлением будет простой линий конвейера. При параллельной конвейеризации время простоя линий сокращается в связи с независимостью работы каждой линии от работы остальных.

Цель работы: получение навыков программирования, тестирования полученного программного продукта и проведения замеров времени выполнения по результатам работы программы на примере конвейерной обработки данных из документов.

Задачи работы:

1) изучение теоретических основ конвейерной обработки данных;
2) описание алгоритмов обработки данных, реализованных на разных этапах конвейера — токенизации текста, установки правил токенизации текста и сортировки токенов в алфавитном порядке;
3) реализация данных алгоритмов;
4) реализация линейных конвейерных вычислений с не менее чем тремя линиями;
5) реализация параллельных конвейерных вычислений с не менее чем тремя линиями;
6) проведение замеров времени работы (в мкс) данных алгоритмов;
7) получение графической зависимости замеряемой величины от количества заявок, предоставляемых на вход конвейеру;
8) проведение сравнительного анализа двух представленных реализаций конвейерных вычислений на основе полученной зависимости.

1. Аналитическая часть

В данном разделе будут рассмотрены теоретические основы линейной и параллельной конвейерной обработки данных и алгоритмы токенизации, применения правил к токенам и их сортировки.

1.1. Конвейерная обработка данных

Конвейер — способ организации вычислений, используемый в современных процессорах с целью повышения их производительности или, в широком смысле, механизм, который позволяет обрабатывать заявки определённого вида поэтапно. На каждой из лент конвейера выполняется одна из стадий обработки поступающих заявок.

Каждая лента в контексте разработки программы — это функция-обработчик, выполняющая над неким набором данных операции и передающая их следующей функции. Под каждую ленту конвейера (функцию-обработчик) может быть выделен отдельный поток, что позволит дополнительно увеличить быстродействие.

1.1.1. Линейная конвейеризация

При линейной конвейеризации одновременно в системе не может находиться более, чем одна заявка: пока одна заявка полностью не пройдёт цикл конвейера, пройденные ей ленты будут простаивать, а другие заявки в очереди на вход в систему будут находиться в состоянии ожидания. При такой реализации время ожидания в очередях между этапами конвейера заявки, уже вошедшей в систему, будет незначительным или отсутствовать вовсе, так как промежуточные очереди не заполняются.

1.1.2. Параллельная конвейеризация

При параллельной конвейеризации одновременно в системе может находиться несколько заявок, если в данный момент времени они проходят обработку на разных этапах конвейера. Таким образом сокращается время ожидания заявок в первичной очереди, однако промежуточные очереди системы будут наполняться заявками, что приведёт к росту времени ожидания заявки в каждой из них.

1.2. Алгоритм токенизации

Токенизация — процесс разделения текста на составляющие (их называют «токенами»). Чаще всего текст разделяется на токены по словам или предложениям. В данной работе принято решение разбивать текст на токены по словам. Отдельные числа и диапазоны чисел будут рассматриваться как один токен.

Для проведения токенизации текст каждого документа просматривается с использованием регулярного выражения ([^0-9а-яА-ЯЁё-]), которое исключает из рассмотрения все последовательности, кроме состоящих из строчных и заглавных букв русского алфавита, цифр и символа «-» для слов, пишущихся через дефис, и числовых диапазонов.

1.3. Алгоритм применения правил к токенам

После токенизации полученный набор токенов просматривается заново, с целью применения установленных правил. В данной работе используются правила объединения нескольких токенов в один, в случае совпадения с заданной конструкцией. Например, комбинация токенов «и», «так», «далее» может быть рассмотрена как один токен «и так далее».

Правила, применяемые к токенам, в свою очередь также составляются по определённой закономерности: любой токен, указанный в правиле, как целевой при замене, должен находиться ещё в одном правиле и преобразовываться сам в себя. Это необходимо, чтобы избежать зацикливания замены, когда токен постоянно будет преобразовываться из одного в другой, и неоднозначности преобразования, в случае если зацикливание всё же не произойдет.

1.4. Алгоритм сортировки токенов по алфавиту

Токены после применения к ним правил сортируются в лексикографическом порядке. Это значит, что при посимвольном сравнении строк большей (при сортировке по возрастанию) будет считаться та, текущий символ которой имеет меньшее значение кода в рассматриваемой кодировке. Для этого применяется алгоритм быстрой сортировки без шаблонов, описанный в [8].

2. Конструкторская часть

В данном разделе будут представлены схемы реализаций алгоритмов конвейерных вычислений, токенизации, применения правил к токенам и сортировки токенов по алфавиту. Также будет приведена общая схема алгоритма функции-обработчика для конвейерных вычислений.

2.1. Разработка алгоритмов конвейерных вычислений

На рисунке 2.1 представлена схема реализации алгоритма линейных конвейерных вычислений и схема реализации алгоритма параллельных конвейерных вычислений, общая схема реализации функции-обработчика представлена на рисунке 2.2.

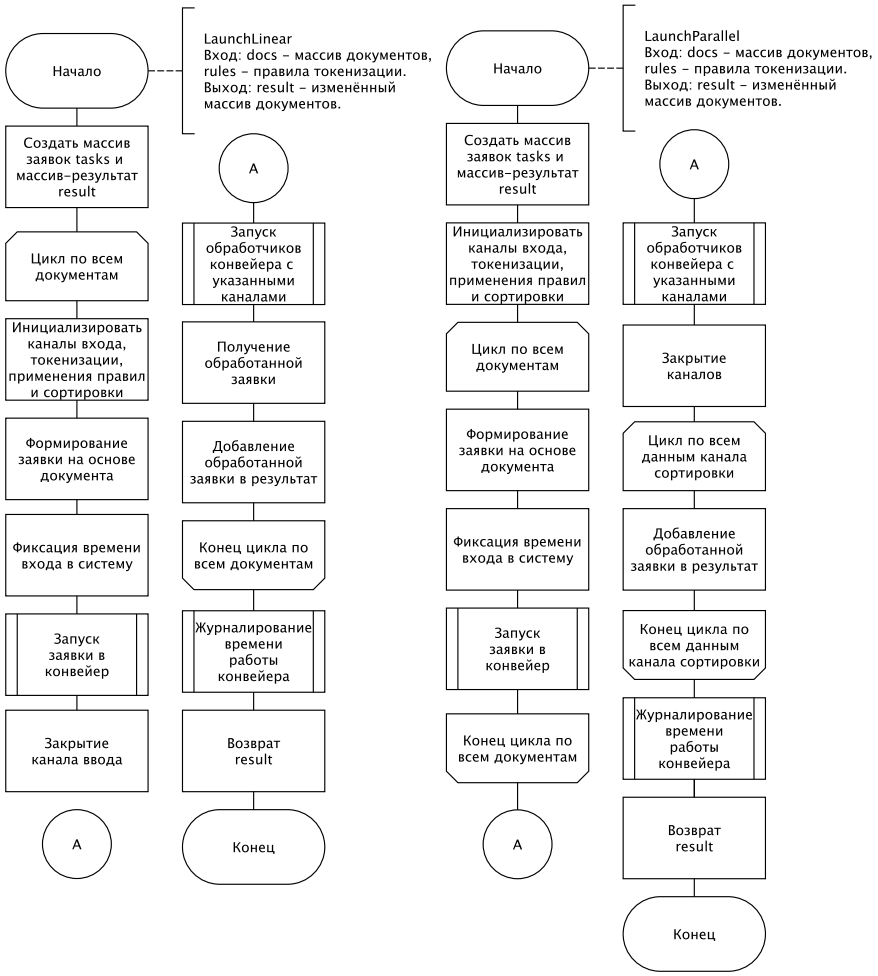

Рисунок 2.1 — Схема реализации алгоритма линейных конвейерных вычислений

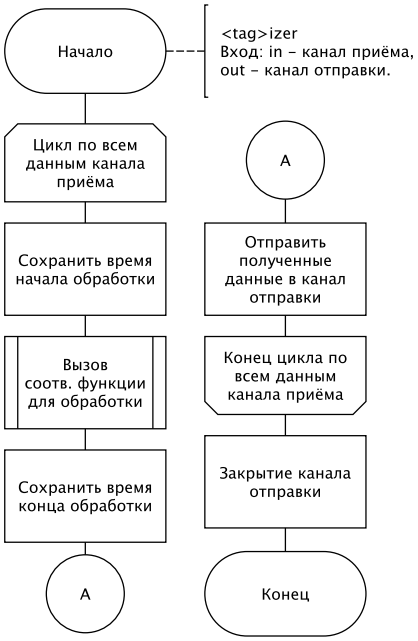

Рисунок 2.2 — Общая схема реализации функции-обработчика для конвейерных вычислений

2.2. Разработка алгоритма токенизации текста

На рисунке 2.3 представлена схема реализации алгоритма токенизации текста.

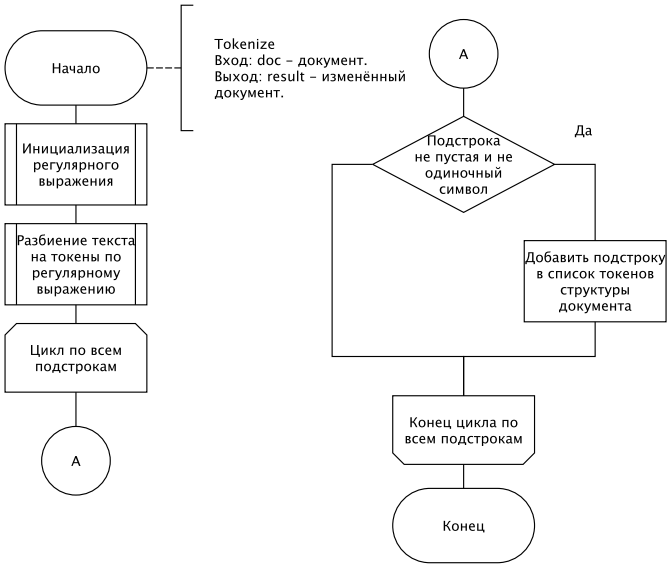

Рисунок 2.3 — Схема реализации алгоритма токенизации текста

2.3. Разработка алгоритма применения правил к токенам текста

На рисунке 2.4 представлена схема реализации алгоритма применения правил к токенам текста.

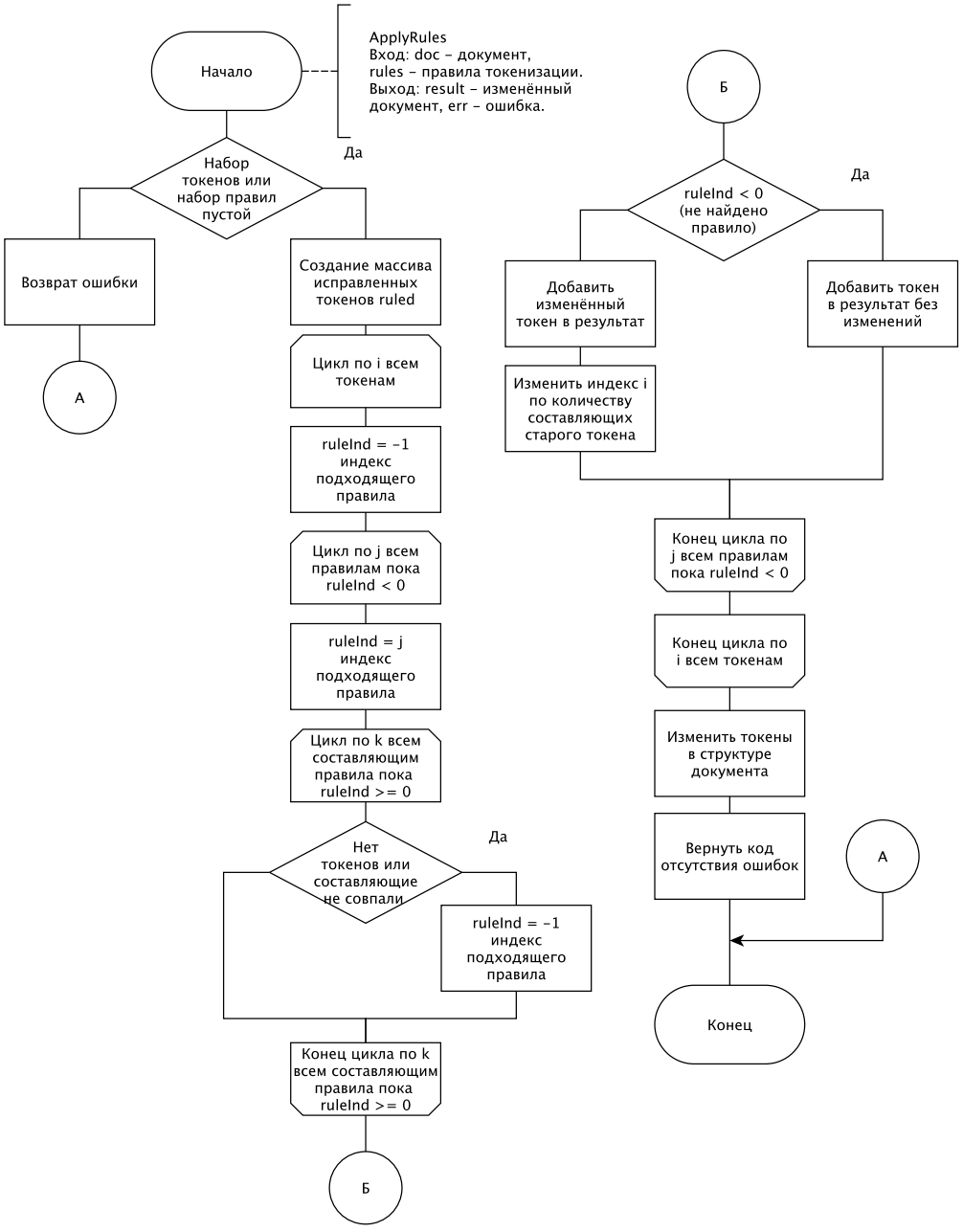

Рисунок 2.4 — Схема реализации алгоритма применения правил к токенам текста

2.4. Разработка алгоритма сортировки токенов по алфавиту

Разработка алгоритма сортировки токенов по алфавиту исключается из рассмотрения в связи с наличием готовых решений, не требующих разработки [8].

3. Технологическая часть

В данном разделе будет представлена реализация алгоритмов конвейерных вычислений, токенизации, применения правил к токенам и сортировки токенов по алфавиту. Также будут указаны обязательные требования к ПО, средства реализации алгоритмов и результаты проведённого тестирования программы.

3.1. Требования к ПО

Для программы выделен перечень требований:

— программой предоставляется интерфейс в формате меню с возможностью ввода пути к директории с обрабатываемыми документами и количества обрабатываемых документов (заявок);

— программой обрабатываются документы, находящиеся в указанной директории;

— программой принимается путь к конфигурационному файлу с правилами токенизации в качестве параметра при запуске;

— программой предлагается повторно выполнить ввод при невалидном выборе пункта меню;

— программой производится аварийное завершение с текстом об ошибке при иных ошибках;

— программой проводится модульное тестирование функций реализации конвейерных вычислений, токенизации, применения правил к токенам;

— программой производятся замеры времени выполнения;

— программой сортируются токены по возрастанию в лексикографическом порядке;

— программой допускается только ввод файлов, текст которых содержит буквы русского алфавита, арабские цифры и знаки препинания.

3.2. Средства реализации

Для реализации данной работы был выбран язык программирования Go [1]. Выбор обусловлен наличием в *Go* библиотек для тестирования ПО и проведения замеров времени выполнения в микросекундах [6][9], а также необходимых для реализации поставленных цели и задач средств. Также, *Go* предоставляет возможность реализовывать потокобезопасные очереди и конвейеры при использовании каналов типа *chan* и параллельных операций *goroutine*, которые могут выполняться независимо от функции, в которой они запущены [10]. В качестве среды разработки была выбрана *GoLand* [3].

3.3. Реализация алгоритмов

В листингах 3.1 – 3.6 представлены реализации функций линейных и параллельных конвейерных вычислений, и подпрограмм токенизации и применения правил к токенам в листингах 3.7 – 3.9.

Листинг 3.1 — Листинг функции линейных конвейерных вычислений (начало)

```
func LaunchLinear(docs []document.Document, rules []rule.Rule)
[]document.Document {
        result := make([]document.Document, 0)
        tasks := make([]Task, 0)

        for _, v := range docs {
                input := make(chan Task, len(docs))
                tokenized := make(chan Task, len(docs))
                ruled := make(chan Task, len(docs))
                sorted := make(chan Task, len(docs))

                task := Task{
                        Doc:       v,
                        TimeFirst: time.Now(),
                }
                input <- task
                close(input)
```

Листинг 3.2 — Листинг функции линейных конвейерных вычислений (окончание листинга 3.1)

```
                tokeniser(input, tokenized)
                ruler(tokenized, ruled, rules)
                sorter(ruled, sorted)

                res := <-sorted

                result = append(result, res.Doc)
                tasks = append(tasks, res)
        }

        LogTasks(tasks)

        return result
}
```

Листинг 3.3 — Листинг функции параллельных конвейерных вычислений (начало)

```
func LaunchParallel(docs []document.Document, rules []rule.Rule)
[]document.Document {
        result := make([]document.Document, 0)
        tasks := make([]Task, 0)
        input := make(chan Task, len(docs))
        tokenized := make(chan Task, len(docs))
        ruled := make(chan Task, len(docs))
        sorted := make(chan Task, len(docs))
        for _, v := range docs {
                task := Task{
                        Doc:       v,
                        TimeFirst: time.Now(),
                }
                input <- task
        }
```

Листинг 3.4 — Листинг функции параллельных конвейерных вычислений (окончание листинга 3.3)

```
        go tokeniser(input, tokenized)
        go ruler(tokenized, ruled, rules)
        go sorter(ruled, sorted)
        close(input)

        for v := range sorted {
                result = append(result, v.Doc)
                tasks = append(tasks, v)
        }
        LogTasks(tasks)
        return result
}
```

Листинг 3.5 — Листинг функций-обработчиков (начало)

```
func tokeniser(in <-chan Task, out chan<- Task) {
        for v := range in {
                v.TimeStart1 = time.Now()
                v.Doc.Tokenize()
                v.TimeEnd1 = time.Now()
                out <- v
        }
        close(out)
}

func ruler(in <-chan Task, out chan<- Task, rules []rule.Rule) {
        for v := range in {
                v.TimeStart2 = time.Now()
                v.Doc.ApplyRules(rules)
                v.TimeEnd2 = time.Now()
                out <- v
        }
        close(out)
}
```

Листинг 3.6 — Листинг функций-обработчиков (окончание листинга 3.5)

```
func sorter(in <-chan Task, out chan<- Task) {
        for v := range in {
                v.TimeStart3 = time.Now()
                sort.Strings(v.Doc.Tokens)
                v.TimeEnd3 = time.Now()
                out <- v
        }
        close(out)
}
```

Листинг 3.7 — Листинг функции токенизации

```
func (doc *Document) Tokenize() {
        doc.Tokens = nil
        re := regexp.MustCompile(`([^0-9a-яA-ЯЁё-])`) //|[+]
        splitted := re.Split(doc.Text, -1)
        for i := range splitted {
                if splitted[i] != "" && splitted[i] != "-" {
                        doc.Tokens = append(doc.Tokens,
                        strings.ToLower(splitted[i]))
                }
        }
}
```

Листинг 3.8 — Листинг функции применения правил к токенам (начало)

```
func (doc *Document) ApplyRules(rules []rule.Rule) error {
        if len(doc.Tokens) == 0 {
                return errors.New("no tokens")
        }
        if len(rules) == 0 {
                return nil
        }
        ruled := make([]string, 0)
```

Листинг 3.9 — Листинг функции применения правил к токенам (окончание листинга 3.8)

```
        for i := 0; i < len(doc.Tokens); i++ {
                ruleInd := -1

                for j := 0; j < len(rules) && ruleInd < 0; j++ {
                        ruleInd = j

                        for k := 0; k < len(rules[j].Option) &&
                        ruleInd >= 0; k++ {
                                if i+k >= len(doc.Tokens) ||
                                doc.Tokens[i+k] != rules[j].Option[k] {
                                        ruleInd = -1
                                }
                        }
                }

                if ruleInd < 0 {
                        ruled = append(ruled, doc.Tokens[i])
                } else {
                        ruled = append(ruled, rules[ruleInd].Standard)
                        i += len(rules[ruleInd].Option) - 1
                }
        }

        doc.Tokens = ruled

        return nil
}
```

3.4. Тестирование

В таблице 3.1 представлены тесты для конвейерной обработки данных. Все тесты пройдены успешно. Документы для тестов либо составлялись вручную, либо копировались со случайных сайтов.

Таблица 3.1 — Тесты для конвейерной обработки данных

№	Количество заявок	Результат
1	0	Сообщение об ошибке
2	-3	Сообщение об ошибке
3	1	Журналирование конвейерных вычислений
4	3	Журналирование конвейерных вычислений
5	8	Журналирование конвейерных вычислений
6	10	Журналирование конвейерных вычислений

4. Экспериментальная часть

В данном разделе описаны проведённые замеры и представлены результаты исследования. Также будут уточнены характеристики устройства, на котором проводились замеры.

4.1. Технические характеристики

Технические характеристики устройства, на котором выполнялось тестирование [4]:

— операционная система *macOS Monterey* 12.4;

— 8 ГБ оперативной памяти;

— процессор *Apple M2* (базовая частота — 2400 МГц, но поддержка технологии *Turbo Boost* позволяет достигать частоты в 3500 МГц [5]).

4.2. Измерение времени выполнения реализаций алгоритмов

Замеры времени работы реализаций алгоритмов производились при помощи встроенных в Go средств, а именно «бенчмарков» (*benchmarks* из пакета *testing* стандартной библиотеки *Go* [1]), представляющих собой тесты производительности [12]. Для построения графика использовалась графическая утилита *gnuplot* [11].

Значение N динамически изменяется для достижения стабильного результата при различных условиях, но гарантируется, что каждый «бенчмарк» будет выполняться хотя бы одну секунду. Для замеров отправлялось количество заявок от 1 до 10, где в качестве исходных данных были тексты, написанные вручную, взятые из книг или со случайных страниц Википедии. Размеры файлов для замеров — до 100000 слов. Результаты тестирования возвращаются в структуре специального вида. Пример такой структуры представлен в листинге 4.10.

Листинг 4.10 — Листинг структуры результата «бенчмарка»

```
testing.BenchmarkResult{N:120000, T:1200000000, Bytes:0, MemAllocs:0x0,
MemBytes:0x0, Extra:map[string]float64{}}
```

В листинге 4.11 представлен пример реализации «бенчмарка».

Листинг 4.11 — Листинг примера реализации «бенчмарка»

```
func NewBenchmarkParallel(docs []document.Document,
rules []rule.Rule) func(*testing.B) {
        return func(b *testing.B) {
                for j := 0; j < b.N; j++ {
                        pipeline.LaunchParallel(docs, rules, 0)
                }
        }
}
```

В таблице 4.1 представлены результаты замеров времени работы конвейера (в мкс.) при разном количестве входных заявок. На рисунке 4.1 приведен график, отражающий зависимость времени выполнения реализаций конвейерных вычислений от количества заявок в очереди. В легендах графиков обозначение *Lin* значит последовательный конвейер, а *Par* — параллельный.

Таблица 4.1 — Результаты замеров времени работы конвейера при разном количестве входных заявок (мкс.)

Кол-во заявок	Линейный конвейер	Параллельный конвейер
1	101 081	101 914
2	197 148	176 099
3	292 401	246 154
4	384 686	319 440
5	481 481	387 951
6	575 756	460 617
7	665 199	528 644
8	767 686	608 424
9	847 215	668 477
10	940 343	741 789

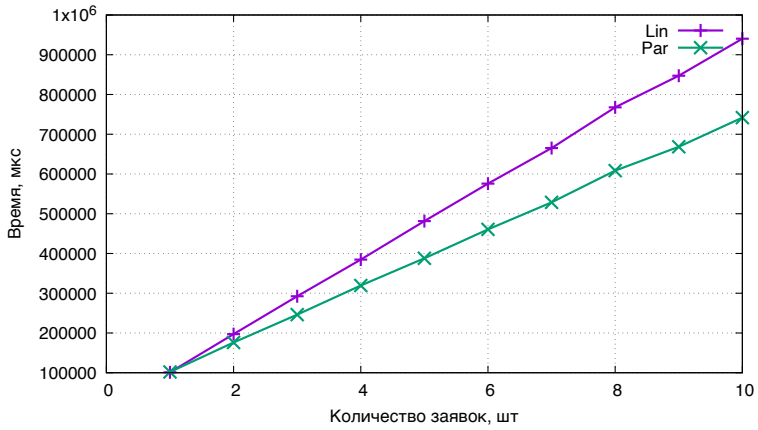

Рисунок 4.1 — Зависимость времени работы конвейера от количества входных заявок

Замеры для журналирования проводились при помощи стандартной библиотеки Go [6][9]. В таблице 4.2 представлен пример журналирования времени обработки заявок для последовательного конвейера, в таблице 4.3 представлен пример журналирования времени обработки заявок для параллельного конвейера, при количестве заявок равном четырём.

Таблица 4.2 — Пример журналирования времени работы последовательного конвейера (мкс.)

№ заявки	Этап	Начало	Конец
1	1	46	71 027
1	2	71 028	77 461
1	3	77 462	85 849
2	1	85 852	129 239
2	2	129 240	134 997
2	3	134 997	142 526
3	1	142 528	185 578
3	2	185 580	190 813
3	3	190 813	198 346
4	1	198 348	241 285
4	2	241 286	247 956
4	3	247 957	255 470

Таблица 4.3 — Пример журналирования времени работы параллельного конвейера (мкс.)

№ заявки	Этап	Начало	Конец
1	1	2	75 155
1	2	75 176	85 541
1	3	85 544	99 337
2	1	75 159	149 655
2	2	149 677	159 719
2	3	159 723	173 595
3	1	149 658	225 378
3	2	225 398	235 442
3	3	235 446	249 523
4	1	225 381	300 991
4	2	300 996	314 670
4	3	314 674	328 010

В таблице 4.4 представлен анализ характеристик работы реализаций алгоритмов конвейеров для количества заявок, равного четырём.

Таблица 4.4 — Таблица конвейерных характеристик

Характеристика		Параллельно, мкс.			Линейно, мкс.		
Линия		1	2	3	1	2	3
Простой очереди	gen	491052	78	13	615105	4	1
	min	100	19	3	37	0	0
	max	240133	20	4	302643	2	1
	avg	122763	19	3	153776	1	0
Время заявки в системе	min	112855			107555		
	max	337209			400358		
	avg	225233			253853		

Заключение

Из проведённых замеров времени работы реализаций последовательной и параллельной конвейерной обработки данных можно сделать следующие выводы. Реализация параллельных конвейерных вычислений выполняется быстрее реализации линейных конвейерных вычислений в 1.26 раза при 8 заявках. При реализации параллельных конвейерных вычислений возникает ситуация, когда на двух линиях простоя в очереди практически нет — общее время простоя на первой линии составляет 491052 микросекунд против 78 микросекунды на второй линии. Это обусловлено тем, что первый обработчик конвейера выполняет обработку данных дольше, чем другие. Среднее время заявки в системе на синхронном конвейере в 1.12 раз больше, чем на асинхронном. Такое поведение является следствием отсутствия простоя во всех очередях, кроме первичной, в случае отсутствия параллельности вычислений. Но, несмотря на это, реализация параллельной конвейерной обработки в целом будет работать быстрее, так как многие вычисления для разных заявок происходят одновременно.

В ходе выполнения лабораторной работы была достигнута поставленная цель: были получены навыки программирования, тестирования полученного программного продукта и проведения замеров времени выполнения и потребляемой памяти по результатам работы программы на примере конвейерной обработки данных из документов.

В процессе выполнения лабораторной работы были также реализованы все поставленные задачи, а именно:

— были изучены теоретические основы конвейерной обработки данных;

— были описаны алгоритмы обработки данных, реализованных на этапах конвейера — токенизации текста, установки правил токенизации текста и сортировки токенов в алфавитном порядке;

— была выполнена программная реализация данных алгоритмов;

— была выполнена программная реализация линейных конвейерных вычислений с не менее чем тремя линиями;

— была выполнена программная реализация параллельных конвейерных вычислений с не менее чем тремя линиями;

— были проведены замеры времени работы (в мкс) для данных алгоритмов;

— была получена зависимость замеряемой величины от количества заявок, предоставляемых на вход конвейеру;

— был проведен сравнительный анализ двух представленных реализаций конвейерных вычислений на основе полученной зависимости.

Список использованных источников

1. Документация по языку программирования *Go* [Электронный ресурс]. Режим доступа: *https://go.dev/doc/* (дата обращения: 20.09.2022).

2. Документация по пакетам языка программирования *Go* [Электронный ресурс]. Режим доступа: *https://pkg.go.dev* (дата обращения: 20.09.2022).

3. GoLand: IDE для профессиональной разработки на *Go* [Электронный ресурс]. Режим доступа: *https://www.jetbrains.com/ru-ru/go/* (дата обращения: 20.09.2022).

4. Техническая спецификация ноутбука *MacBook Air* [Электронный ресурс]. Режим доступа: *https://support.apple.com/kb/SP869* (дата обращения: 20.09.2022).

5. *Apple M2* [Электронный ресурс]. Режим доступа: *https://www.notebookcheck.net/Apple-M2-Processor-Benchmarks-and-Specs.632312.0.html* (дата обращения: 10.10.2022).

6. Документация по пакету *time* стандартной библиотеки *Go* [Электронный ресурс]. Режим доступа: *https://pkg.go.dev/time* (дата обращения: 10.10.2022).

7. Allan V. H. et al. Software pipelining //ACM Computing Surveys (CSUR). – 1995. – Т. 27. – №. 3. – С. 367-432.

8. *Pattern – defeating Quicksort* [Электронный ресурс]. Режим доступа: *https://arxiv.org/pdf/2106.05123.pdf* (дата обращения: 10.12.2022).

9. Документация по методу *Duration.Microseconds* пакета *time* стандартной библиотеки *Go* [Электронный ресурс]. Режим доступа: *https://pkg.go.dev/time#Duration.Microseconds* (дата обращения: 10.10.2022).

10. Donovan A. A. A., Kernighan B. W. The Go programming language. – Addison-Wesley Professional, 2015.

11. *gnuplot homepage* [Электронный ресурс]. Режим доступа: *http://www.gnuplot.info* (дата обращения: 10.10.2022).

12. Исходный код *src/testing/benchmark.go* [Электронный ресурс]. Режим доступа: *https://go.dev/src/testing/benchmark.go* (дата обращения: 10.10.2022).