



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»

«Параллельное программирование»

Студент _____ Маслова Марина Дмитриевна

Группа _____ ИУ7-53Б

Оценка (баллы) _____

Преподаватель _____ Волкова Лилия Леонидовна

2021 г.

Содержание

Введение	3
1 Аналитическая часть	4
1.1 Конвейерная обработка данных	4
1.2 Стадии конвейерной обработки	4
1.2.1 Генерация сообщения	5
1.2.2 Перестановка букв в словах в обратном порядке	5
1.2.3 Шифр Веженера	5
1.3 Вывод	5
2 Конструкторская часть	7
2.1 Разработка алгоритмов	7
2.2 Структура разрабатываемого ПО	7
2.3 Классы эквивалентности при тестировании	7
2.4 Вывод	7
3 Технологическая часть	9
3.1 Требования к программному обеспечению	9
3.2 Средства реализации	9
3.3 Листинги кода	10
3.4 Описание тестирования	13
3.5 Вывод	13
4 Исследовательская часть	14
4.1 Технические характеристики	14
4.2 Примеры работы программы	14
4.3 Результаты тестирования	14
4.4 Постановка эксперимента по замеру времени	14
4.5 Результаты эксперимента	16
4.6 Вывод	18
Заключение	19
Список литературы	20

Введение

Сегодня программирование используется во многих научных и социальных областях. Компьютерам требуется производить все более трудоемкие вычисления на больших объемах данных. При этом предъявляются требования к скорости вычислений.

При обработке больших объемов данных часто возникает ситуация, когда каждый набор данных необходимо обработать последовательно несколькими алгоритмами. Так как каждый набор может быть обработан независимо от другого имеется возможность распараллелить вычисления. В таком случае релизуют конвейерную обработку данных, когда каждый алгоритм выполняется на отдельной "ленте", запущенной в отдельном потоке, а результат каждого этапа используется в качестве входных данных следующего.

Целью данной работы является получение навыков конвейерной обработки данных.

Для достижения поставленной цели необходимо выполнить следующие **задачи**:

- описать конвейерную обработку данных;
- описать алгоритмы, реализуемые на каждом этапе конвейера;
- описать функциональные требования;
- разработать описанные алгоритмы;
- реализовать алгоритмы всех этапов;
- реализовать конвейерную обработку данных;
- реализовать линейную обработку данных;
- провести тестирование реализованных алгоритмов;
- провести сравнительный анализ алгоритмов по времени работы реализаций;
- сделать выводы по полученным результатам.

1 Аналитическая часть

В данном разделе представлено теоретическое описание конвейерной обработки данных и алгоритмов каждой стадии обработки.

1.1 Конвейерная обработка данных

Конвейерная обработка данных – это подход, позволяющий создавать эффективные параллельные реализации обычных неспециализированных алгоритмов [1]. Общая идея конвейера связана с разбиением некоторого процесса обработки объектов на несколько независимых этапов и организацией одновременного (параллельного) выполнения этих этапов обработки различных объектов, передвигающихся по конвейеру от одного этапа к другому. При движении объектов по конвейеру на разных его участках выполняются разные операции, а при достижении каждым объектом конца конвейера он окажется полностью обработанным [2].

При реализации конвейерной обработки каждый из этапов должен [1]:

1. получить данные;
2. обработать данные;
3. передать данные следующим этапам.

В данной лабораторной работе будет реализован конвейер, состоящий из трех этапов обработки данных, каждый из которых будет реализован в отдельном потоке. Передача данных между потоками будет реализована с помощью четырех очередей:

- от главного потока к первому этапу;
- от первого этапа ко второму этапу;
- от второго этапа к третьему этапу;
- от третьего этапа к главному потоку.

1.2 Стадии конвейерной обработки

Для проведения исследования в данной лабораторной работе на конвейере будут генерироваться зашифрованные сообщения. На первой ленте будет генерироваться случайное сообщение на английском языке. На второй ленте

каждое слово в сообщении будет записываться в обратном порядке. На третьей ленте к сообщению будет применен шифр Веженера.

Далее описываются алгоритмы, выполняющиеся на каждой ленте конвейера.

1.2.1 Генерация сообщения

На первой ленте генерируются случайные сообщения на английском языке и переводятся в нижний регистр для последующей неразличимости первых слов предложений и имен собственных.

1.2.2 Перестановка букв в словах в обратном порядке

На второй ленте у каждого слова сгенерированного на первом этапе сообщения переставляются буквы в обратном порядке.

1.2.3 Шифр Веженера

На третьей ленте полученное на втором этапе сообщение зашифровывается с помощью шифра Веженера.

Шифр Веженера является частным случаем многоалфавитной замены. Формально данный алгоритм шифрования можно описать следующим образом. Выбирается ключ шифрования – набор из m целых чисел $k = (k_1, k_2, \dots, k_m)$ (обычно выбирается какое-либо слово, буквам которого ставится в соответствие число – порядковый номер в алфавите) [3]. Каждый символ зашифрованного сообщения c_i получается из соответствующего символа t_i данного текста по следующей формуле:

$$c_i = t_i + k_i \bmod m+1 \pmod{|V|}, \quad (1.1)$$

где $i \in \{1, 2, \dots, n\}$ (n – длина текста), $|V|$ – мощность алфавита.

1.3 Вывод

В данном разделе была описана модель конвейерной обработки данных, также были представлены описания алгоритмов, выполняющихся на каждой

ленте конвейера. В качестве реализуемого поэтапного алгоритма была выбрана генерация зашифрованных сообщений. Из представленных описаний можно предъявить ряд требований к разрабатываемому программному обеспечению:

- на вход должно подаваться количество обрабатываемых заявок;
- при неверных входных данных должно выдаваться сообщение об ошибке;
- на выходе должны выдаваться логи работы каждого из методов обработки (последовательного и конвейерного).

2 Конструкторская часть

В данном разделе разрабатываются последовательный и параллельный алгоритмы, численного интегрирования методом средних прямоугольников, структура программы и способы её тестирования.

2.1 Разработка алгоритмов

На рисунках ??-?? представлена схема алгоритма последовательного алгоритма численного интегрирования методом средних прямоугольников с заданной точностью, на рисунках ??-?? — схема параллельного алгоритма.

2.2 Структура разрабатываемого ПО

Для реализации разрабатываемого программного обеспечения будет использоваться метод структурного программирования. Каждый из алгоритмов будет представлен отдельной функцией, при необходимости будут выделены подпрограммы для каждой из них. Также будут реализованы функции для ввода-вывода и функция, вызывающая все подпрограммы для связности и полноценности программы.

2.3 Классы эквивалентности при тестировании

Для тестирования программного обеспечения во множестве тестов будут выделены следующие классы эквивалентности:

- совпадение верхнего и нижнего пределов интегрирования;
- положительные пределы интегрирования;
- верхний предел интегрирования меньше нижнего предела;
- произвольные пределы интегрирования.

2.4 Вывод

В данном разделе были разработаны последовательный и параллельный алгоритмы, была описана структура разрабатываемого ПО. Для дальнейшей

проверки правильности работы программы были выделены классы эквивалентности тестов.

3 Технологическая часть

В данном разделе описаны требования к программному обеспечению, средства реализации, приведены листинги кода и данные, на которых будет проводиться тестирование.

3.1 Требования к программному обеспечению

Программа должна предоставлять следующие возможности:

- ввода количества обрабатываемых заявок;
- вывода лога работы программы для введенного количества заявок, включающих для каждой заявки время начала и конца обработки на каждой из лент;
- получения времени обработки всех заявок.

3.2 Средства реализации

Для реализации данной лабораторной работы выбран компилируемый многопоточный язык программирования Go[4], так как он предоставляет необходимый функционал для работы с потоками и простой реализации их взаимодействия. Интерпретируемый язык программирования высокого уровня Python[5] был выбран для визуализации данных эксперимента, так как он предоставляет большое число настроек параметров графика с использованием простого синтаксиса.

В качестве среды разработки выбран текстовый редактор Vim[6] с установленными плагинами автодополнения и поиска ошибок в процессе написания, так как он реализует быстрое перемещение по тексту программы и простое взаимодействие с командной строкой.

Замеры времени проводились при помощи функции `Now()` из библиотеки `time`[7].

3.3 Листинги кода

В данном подразделе представлены листинги кода алгоритмов:

- реализация генерации сообщений (листинг 3.1);
- реализация перестановки букв в каждом слове в обратном порядке (листинг 3.2);
- реализация шифра Верженера (листинг 3.3);
- последовательная реализация конвейера (листинг 3.4);
- параллельная реализация конвейера (листинг 3.5).

Листинг 3.1 – Функция генерации сообщений

```
1 func GenerateMsg(msg *Cipher) {  
2     msg.Msg = gofakeit.Sentence(msg.wordsNum)  
3     msg.Msg = strings.TrimRight(strings.ToLower(msg.Msg), ".")  
4 }
```

Листинг 3.2 – Функции перестановки букв в каждом слове в обратном порядке

```
1 func reverseString(str string) (res string) {  
2     for _, ch := range str {  
3         res = string(ch) + res  
4     }  
5  
6     return res  
7 }  
8  
9 func ReverseWords(msg *Cipher) {  
10    words := strings.Fields(msg.Msg)  
11  
12    for i, word := range words {  
13        words[i] = reverseString(word)  
14    }  
15  
16    msg.Msg = strings.Join(words, " ")  
17 }
```

Листинг 3.3 – Функция шифра Верженера

```
1 func CodeByVegenere(msg *Cipher) {  
2     key := []rune(gofakeit.Word())  
3     keyInd := 0  
4     res := ""
```

Листинг 3.3 (продолжение)

```
5
6     for _, ch := range msg.Msg {
7         if ch >= 'a' && ch <= 'z' {
8             ch -= 'a'
9             keyCh := key[keyInd] - 'a'
10
11             ch = (ch+keyCh)%26 + 'a'
12
13             keyInd++
14             keyInd %= len(key)
15         }
16
17         res += string(ch)
18     }
19
20     msg.Msg = res
21 }
```

Листинг 3.4 – Последовательная обработка заявок

```
1 func LinearConveyor(inQueue *Queue) *Queue {
2     res := CreateQueue(inQueue.capacity)
3
4     c := inQueue.Pop()
5     for c != nil {
6         c.startMsg = time.Now()
7         GenerateMsg(c)
8         c.endMsg = time.Now()
9
10        c.startReplace = time.Now()
11        ReverseWords(c)
12        c.endReplace = time.Now()
13
14        c.startVegenere = time.Now()
15        CodeByVegenere(c)
16        c.endVegenere = time.Now()
17
18        res.Push(c)
19        c = inQueue.Pop()
20    }
21
22    return res
23 }
```

Листинг 3.5 – Конвейерная обработка заявок

```
1 func ParallelConveyor(inQueue *Queue) *Queue {
2     mainToF := make(chan *Cipher, inQueue.capacity)
3     fToS := make(chan *Cipher, inQueue.capacity)
4     sToT := make(chan *Cipher, inQueue.capacity)
5     tToMain := make(chan int)
6     res := CreateQueue(inQueue.capacity)
7
8     first := func() {
9         for {
10             select {
11                 case c := <-mainToF:
12                     c.startMsg = time.Now()
13                     GenerateMsg(c)
14                     c.endMsg = time.Now()
15
16                     fToS <- c
17             }
18         }
19     }
20
21     second := func() {
22         for {
23             select {
24                 case c := <-fToS:
25                     c.startReplace = time.Now()
26                     ReverseWords(c)
27                     c.endReplace = time.Now()
28
29                     sToT <- c
30             }
31         }
32     }
33
34
35     third := func() {
36         for {
37             select {
38                 case c := <-sToT:
39                     c.startVegenere = time.Now()
40                     CodeByVegenere(c)
41                     c.endVegenere = time.Now()
42
43                     res.Push(c)
44                     if res.tail == res.capacity-1 {
45                         tToMain <- 0
```

Листинг 3.5 (продолжение)

```
46         }
47     }
48 }
49 }
50
51 go first()
52 go second()
53 go third()
54
55 cip := inQueue.Pop()
56 for cip != nil {
57     mainToF <- cip
58     cip = inQueue.Pop()
59 }
60
61 <-tToMain
62
63 return res
64 }
```

3.4 Описание тестирования

В таблице 3.1 приведены функциональные тесты программы.

Таблица 3.1 – Функциональные тесты

Количество заявок	Ожидаемый результат
-1	Сообщение об ошибке
0	Сообщение об ошибке
j	Сообщение об ошибке
10	Лог работы каждого из алгоритмов
100	Лог работы каждого из алгоритмов
1000	Лог работы каждого из алгоритмов

3.5 Вывод

В данном разделе были реализованы последовательный и конвейрный алгоритмы поэтапной генерации зашифрованных сообщений. Также были написаны тесты для каждого класса эквивалентности, описанного в конструкторском разделе.

4 Исследовательская часть

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялось тестирование:

- Операционная система: Manjaro [8] Linux x86_64.
- Память: 8 GiB.
- Процессор: Intel® Core™ i5-8265U, 4 физических ядра, 8 логических ядра[9].

Тестирование проводилось на ноутбуке, включенном в сеть электропитания. Во время тестирования ноутбук был нагружен только встроенными приложениями окружения, окружением, а также непосредственно системой тестирования.

4.2 Примеры работы программы

На рисунке 4.1 представлены результаты работы последовательной обработки, на рисунке 4.2 – конвейерной.

4.3 Результаты тестирования

Программа была протестирована на входных данных, приведенных в таблице 3.1. Полученные результаты работы программы совпали с ожидаемыми результатами.

4.4 Постановка эксперимента по замеру времени

Для оценки времени работы последовательной и конвейерной реализации алгоритмов шифрования был проведен эксперимент, в котором определялось влияние количества заявок и количества слов в сообщении на время работы каждого из алгоритмов. Тестирование проводилось на количестве заявок от 5 до 10 с шагом 5, от 25 до 100 с шагом 25 и от 100 до 1000 с

Введите количество заявок: 10

№	STAGE	BEGIN	END
0	1	0s	589.526µs
0	2	589.708µs	1.139813ms
0	3	1.140001ms	1.157167ms
1	1	1.157428ms	1.565489ms
1	2	1.56565ms	2.015814ms
1	3	2.016008ms	2.030578ms
2	1	2.030864ms	2.593928ms
2	2	2.59408ms	3.140298ms
2	3	3.140468ms	3.15836ms
3	1	3.158549ms	3.708474ms
3	2	3.708621ms	4.197549ms
3	3	4.19771ms	4.210166ms
4	1	4.210331ms	4.455225ms
4	2	4.455367ms	4.700676ms
4	3	4.700825ms	4.708093ms
5	1	4.708256ms	4.891977ms
5	2	4.892118ms	5.076471ms
5	3	5.076612ms	5.086448ms
6	1	5.086612ms	5.341224ms
6	2	5.341364ms	5.587944ms
6	3	5.588099ms	5.610904ms
7	1	5.610989ms	5.77255ms
7	2	5.772628ms	5.933782ms
7	3	5.933858ms	5.938324ms
8	1	5.938423ms	6.162889ms
8	2	6.16296ms	6.388129ms
8	3	6.388208ms	6.396652ms
9	1	6.396731ms	6.4935ms
9	2	6.493574ms	6.590821ms
9	3	6.590898ms	6.594015ms
TOTAL:		6.595385ms	

Рисунок 4.1 – Лог последовательной обработки

№	STAGE	BEGIN	END
0	1	0s	258.699µs
1	1	262.71µs	455.463µs
0	2	331.587µs	629.447µs
2	1	455.666µs	712.942µs
1	2	633.75µs	833.271µs
3	1	713.282µs	970.184µs
0	3	759.318µs	815.069µs
2	2	836.839µs	1.094986ms
1	3	916.683µs	923.84µs
4	1	970.521µs	1.09914ms
3	2	1.097857ms	1.355662ms
5	1	1.099472ms	1.196168ms
6	1	1.196382ms	1.325318ms
2	3	1.205403ms	1.241865ms
7	1	1.325507ms	1.486346ms
4	2	1.358054ms	1.501138ms
3	3	1.469222ms	1.480175ms
8	1	1.486636ms	1.711306ms
5	2	1.501788ms	1.604177ms
4	3	1.575251ms	1.582657ms
6	2	1.604578ms	1.739332ms
5	3	1.682687ms	1.68953ms
9	1	1.711585ms	1.811051ms
7	2	1.74278ms	1.909263ms
6	3	1.816496ms	1.824386ms
8	2	1.911683ms	2.1422ms
7	3	2.02037ms	2.028946ms
9	2	2.145408ms	2.242736ms
8	3	2.209746ms	2.216469ms
9	3	2.244407ms	2.250412ms
TOTAL:		2.27347ms	

Рисунок 4.2 – Лог конвейерной обработки

шагом 250, а количество слов принимало значения 5, 10, 25, 50, 75, 100. Время работы на каждом из значений было получено с помощью *бенчмарков*[10], являющимися встроенными средствами языка Go. В них количество повторов измерений времени изменяется динамически до тех пор, пока не будет получен стабильный результат.

Результаты эксперимента были представлены в виде таблиц и графиков, приведенных в следующем подразделе.

4.5 Результаты эксперимента

В таблице 4.1 представлены результаты измерения времени работы последовательной и конвейерной реализаций в зависимости от числа заявок. На рисунке 4.3 представлен соответствующий график.

В таблице 4.2 представлены результаты измерения времени работы последовательной и конвейерной реализаций в зависимости от числа слов в сообщениях при фиксированном числе заявок, равном 50. На рисунке 4.4 представлен соответствующий график.

Таблица 4.1 – Время работы от числа заявок

Число заявок	Последовательная, нс	Конвейерная, нс
5	1398118	1744339
10	2842780	2770063
25	6987262	5289069
50	13423468	8961571
75	20151882	13339268
100	27811972	17249171
250	71697350	40135084
500	133232370	84669318
750	209148024	127482157
1000	276952383	160373144

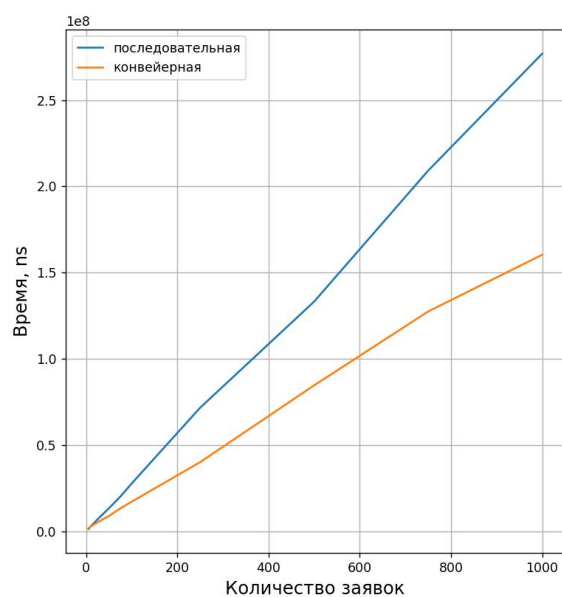


Рисунок 4.3 – График зависимости времени работы от числа заявок

Таблица 4.2 – Время работы от количества слов

Точность	Последовательная, нс	Параллельная, нс
5	12787683	8451608
10	25006741	14918758
25	62884400	37303679
50	132833197	73853581
75	201132505	108303512
100	259405110	143977382

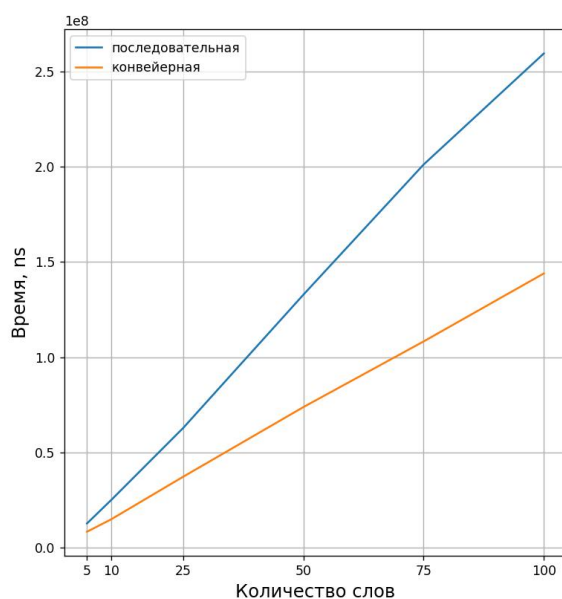


Рисунок 4.4 – График зависимости времени работы реализаций от количества слов в сообщениях

4.6 Вывод

По результатам эксперимента можно сделать следующие выводы:

- при количестве заявок до 10 последовательная и конвейерная реализация генерации зашифрованных сообщений отрабатывают за одинаковое время, а при количестве заявок до 5 последовательная реализация работает 1.3 раза быстрее, что объясняется затратами на передачу данных между лентами с помощью очередей/каналов в конвейерной реализации;
- при большем количестве заявок от 25 конвейерная реализация работает в 1.7 раза быстрее последовательной;
- при фиксированном количестве заявок при различных количествах слов в сообщениях конвейерная реализация работает в 1.7 раза быстрее последовательной.

Таким образом, для обработки количества заявок большего 10 для достижения оптимальной скорости вычислений необходимо использовать конвейерную реализацию. Если работа просходит с количеством заявок до 10 достаточно последовательной реализации, то есть нет необходимости реализовывать более сложный конвейерный алгоритм.

Заключение

В ходе исследования был проведен сравнительный анализ последовательной и конвейерной реализации алгоритма поэтапного шифрования сообщений. В результате исследования было выяснено, что при количестве заявок более 10 конвейерная реализация дает выигрыш в 1.7 раза по сравнению с последовательной, что говорит о преимуществе параллельной реализации этапов конвейера при решении поставленной задачи.

В ходе выполнения лабораторной работы:

- были описаны и разработаны алгоритмы этапов конвейерной обработки данных;
- были описаны и разработаны линейная и конвейерная обработки данных;
- был реализован каждый из описанных алгоритмов;
- по экспериментальным данным были сделаны выводы об эффективности по времени каждого из реализованных алгоритмов;
- были получены зависимости времени работы линейной и конвейерной реализаций от числа и размера заявок.

Таким образом, все поставленные задачи были выполнены, а цель достигнута.

Список литературы

- [1] Применение конвейерной обработки данных на примере сортировки простыми вставками / Д. А. Погорелов, А. М. Таразанов, Л. Л. Волкова [и др.] // Образование и наука в России и за рубежом. 2019. Т. 49, № 1.
- [2] Богословский Н. Н. Конспект лекций дисциплины «Архитектура и программное обеспечение высокопроизводительных вычислительных систем». Режим доступа: http://hpc-education.ru/files/lectures/2011/bogoslovskiy/bogoslovskiy_2011_lectures05.pdf (дата обращения: 08.12.2021).
- [3] Адигеев М. Г. Введение в криптографию. Ростов-на-Дону: Издательство РГУ, 2002.
- [4] The Go Programming Language [Электронный ресурс]. Режим доступа: <https://golang.org/> (дата обращения: 07.12.2021).
- [5] Welcome to Python [Электронный ресурс]. Режим доступа: <https://www.python.org> (дата обращения: 12.10.2021).
- [6] welcome home : vim online [Электронный ресурс]. Режим доступа: <https://www.vim.org/> (дата обращения: 12.10.2021).
- [7] Package time [Электронный ресурс]. Режим доступа: <https://golang.org/pkg/time/> (дата обращения: 07.12.2021).
- [8] Manjaro — enjoy the simplicity [Электронный ресурс]. Режим доступа: <https://manjaro.org/> (дата обращения: 17.10.2021).
- [9] Процессор Intel® Core™ i5-8265U [Электронный ресурс]. Режим доступа: <https://ark.intel.com/content/www/ru/ru/ark/products/149088/intel-core-i5-8265u-processor-6m-cache-up-to-3-90-ghz.html> (дата обращения: 17.10.2021).

- [10] Testing – The Go Programming Language [Электронный ресурс]. Режим доступа: <https://golang.org/pkg/testing/> (дата обращения: 07.12.2021).