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

Название: Конвеерные вычисления

Дисциплина: Анализ алгоритмов

Студент ИУ7-52Б _____ Н. В. Ляпина
(Группа) (Подпись, дата) (И.О. Фамилия)

Преподователь _____ Л.Л. Волкова
(Подпись, дата) (И.О. Фамилия)

Москва, 2022

Содержание

Введение 3
1 Аналитический раздел 4
1.1 Конвеерные вычисления 4
1.2 Шифр Цезаря 4
1.3 XOR-шифр 4
2 Констукторский раздел 6
2.1 Описание структур данных 6
2.2 Схемы алгоритмов 6
2.3 Структура ПО 10
3 Технологический раздел 11
3.1 Требования к ПО 11
3.2 Средства реализации 11
3.3 Листинг программы 11
3.4 Тестирование ПО 15
4 Экспериментальный раздел 17
4.1 Технические характеристики 17
4.2 Описание системы тестирования 17
4.3 Постановка эксперимента 17
4.4 Результаты эксперимента 17
4.5 Вывод 19
Заключение 20
Список источников 21

Введение

Время работы является одной из основных характеристик, которые влияют на оценку того или иного алгоритма. У большинства алгоритмов можно найти модифицированные аналоги, которые стараются улучшить эту характеристику. Сделать алгоритм более быстрым может не только его модификация, но и такой способ как организация вычислительного конвеера.

Метод вычислительного конвеера предполагает декомпозицию задачи на подзадачи таким образом, что результат работы одной подзазачи является входными данными для следующей подзадачи. Таком образом реализуются конвеерные вычисления [1].

В представленной работе исследуется реализация вычислительного конвеера, который используется в качестве декомпозированных подзадач алгоритмы шифрования строк: шифр Цезаря и *xor*-шифр.

Цель лабораторной работы:
изучить и реализовать конвеерные вычисления.

Задачи лабораторной работы:

1) изучить организацию конвеерной обработки данных;
2) изучить шифр Цезаря и *xor*-шифр;
3) реализовать:
    а) последовательную конвеерную обработку данных;
    б) параллельную конвеерную обработку данных.
4) провести замеры процессорного времени работы для реализованных алгоритмов.

1 Аналитический раздел

1.1 Конвеерные вычисления

Способ организации процесса в качетсве вычислительного конвеера позволяет построить процесс, содержащий несколько независимых этапов [1], на нескольких потоках. Выигрыш по времени достигается при выполнении нескольких этапов одновременно за счет параллельной работы ступеней. Для контроля стадии используется три основные метрики, описанные ниже.

1) **Время процесса** – это время, необходимое для выполнения одной стадии;

2) **Время выполнения** – это время, которое требуется с момента, когда работа была выполнена на предыдущем этапе, до выполнения на текущем;

3) **Время простоя** – это время, когда никакой работы не происходит и линии простаивают.

Для того, чтобы время простоя было минимальным, стадии обработки должны быть одинаковы по времени в пределах погрешности. При возникновении ситуации, в которой время процесса одной из линий больше, чем время в других в N раз, эту линию стоит распаралеллить на N потоков.

1.2 Шифр Цезаря

Шифр цезаря – это преобразование информации методом замены букв на другие, стоящии от даннх через определенное количество символов в алфавите. Сдвиг на определенное количество символов называется ключом шифрования.[2]

Таким образом, алгоритм выглядит следующим образом:

1) для каждой буквы исходной строки найти индекс i в алфавите;

2) добавить в результирующую строку букву с индексом $(i + key)\%len(alphabet)$ в алфавите.

1.3 XOR-шифр

XOR-шифрование – это применение ключа через побитовое исключающее ИЛИ к исходному тексту.

Представлена таблица истинности побитового исключающего ИЛИ:

Таким образом, при выполнении исключающего ИЛИ всегда будет нулевое значени, если переменные имели одинаковые значения, иначе будет единица.

Особенность XOR в том, что одной и той же функцией можно как зашифровать данные, так и расшифровать их. Это простой метод шифрации данных, который может быть взломан достаточно быстро при наличии достаточно большого зашифрованного текста, или большого словаря паролей. Но тем не менее это уже можно применять для небольшой первоначально защиты данных [3].

Таблица 1.1 — Таблица истинности побитового исключающего ИЛИ

X	Y	XOR(X, Y)
0	0	0
0	1	1
1	0	1
1	1	0

Вывод

В качестве входных данных для конвеера с тремя линиями достаточно использовать размер очереди, обрабатываемой конвейером.

Конвейер может быть реализован следующим образом:

Линия 1. Шифрование строки XOR-шифром с первым ключом;

Линия 2. Шифрование строки шифром Цезаря;

Линия 3. Шифрование строки XOR-шифром со вторым ключом;

2 Констукторский раздел

В данном разделе будут рассмотрены схемы алгоритмов, требования к функциональности ПО, и определены способы тестирования.

2.1 Описание структур данных

В данной работе линии вычислительного конвейера реализованы с помощью очереди [4], реализованной на одномерном динамическом массиве. Выбор типа данных обусловлен тем, что заявки обрабатываются в порятке их поступления, выполнив их последовательно.

2.2 Схемы алгоритмов

Ниже будут представлены схемы алгоритмов:

1) работы конвейера (рисунок 2.1);
2) XOR-шифра (рисунок 2.2);
3) шифра Цезаря (рисунок 2.3).

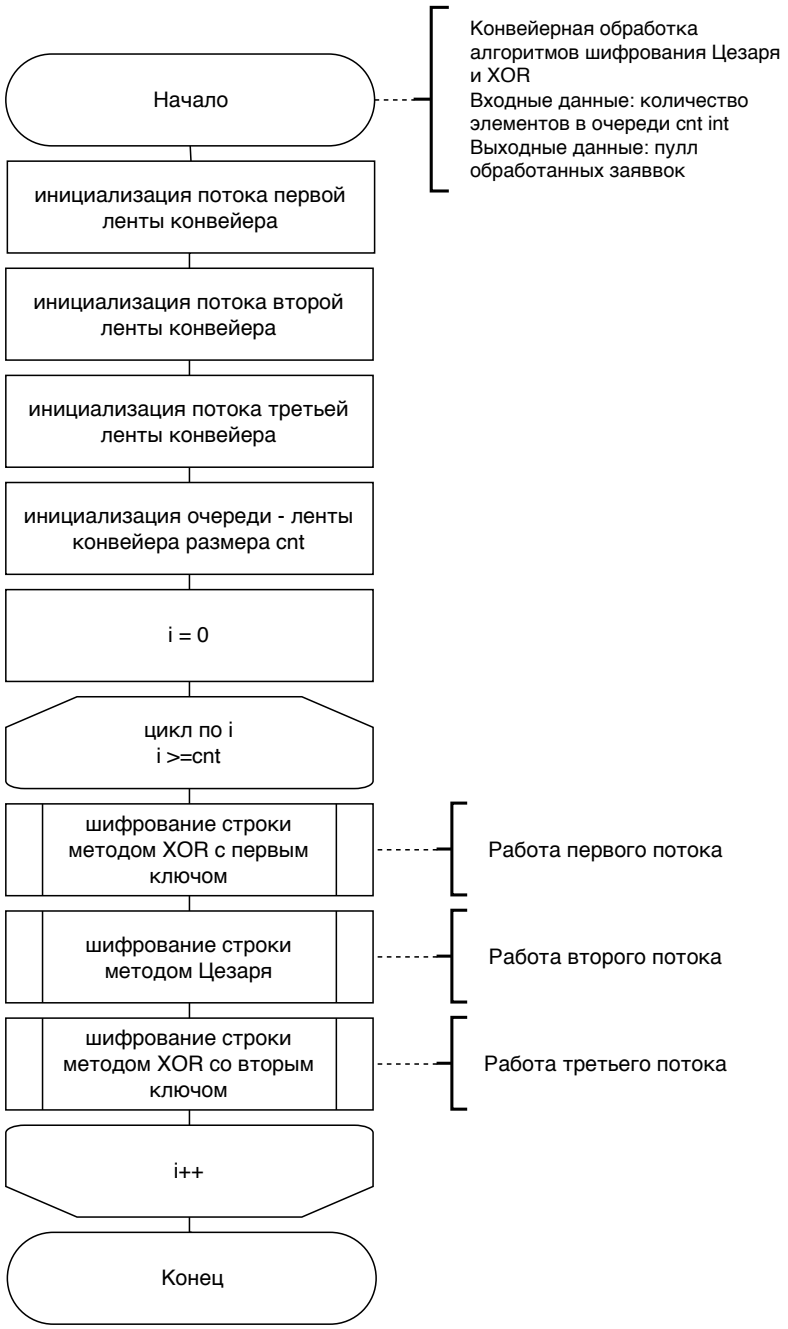

Рисунок 2.1 — Схема работы конвейера

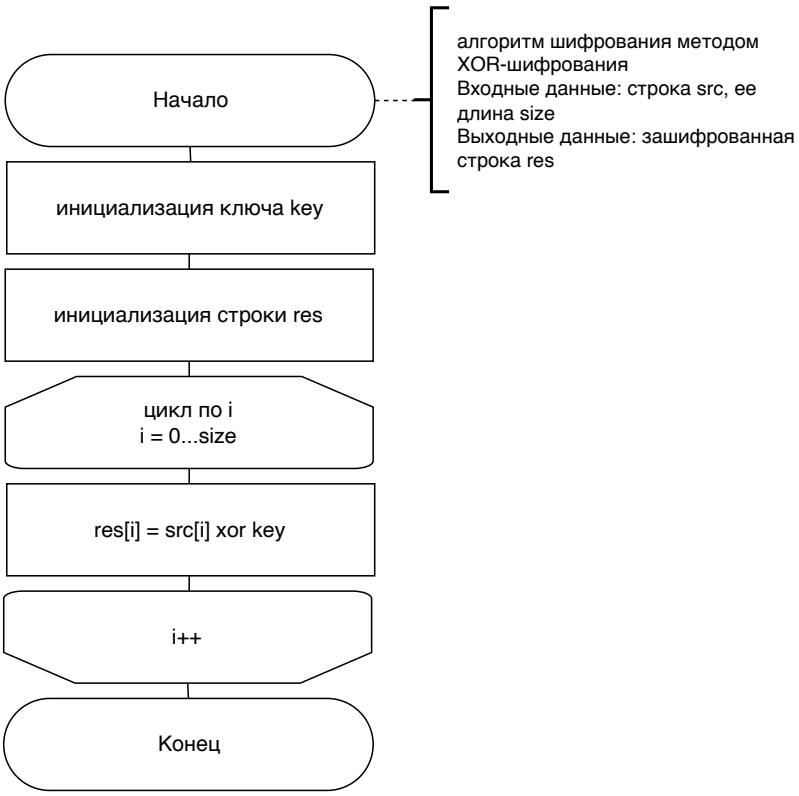

Рисунок 2.2 — Схема алгоритма XOR-шифра

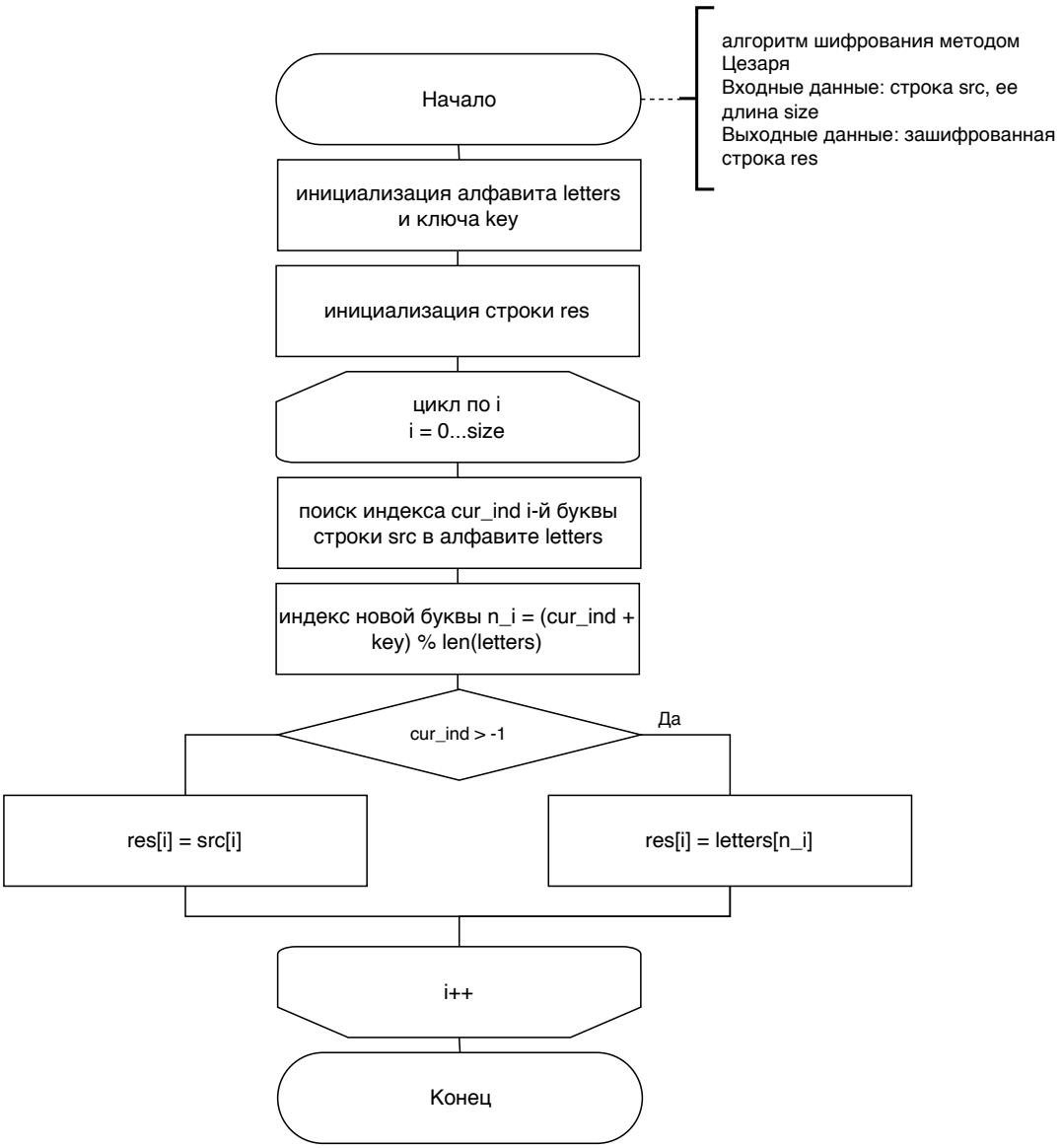

Рисунок 2.3 — Схема алгоритма шифрования методом Цезаря

2.3 Структура ПО

Программа поделена на ряд смысловых модулей, описанных ниже:

— Модуль «chiperalg», в котором содержатся процедуры и функции, связанные с алгоритмами шифрования и генерации строк;

— Модуль «pipeline», в котором содержатся функции работы параллельного и синхронного конвейера, функции работы с очередью и журналирование.

Программа имеет консольный интерфейс.

Вывод

Были разработаны схемы алгоритмов, необходимых для решения задачи. Получено достаточно теоретической информации для написания программного обеспечения.

3 Технологический раздел

В данном разделе представлены средства, использованные в процессе разработки для реализации задачи, а также листинг кода программы. Кроме того показаны результаты тестирования.

3.1 Требования к ПО

Программное обеспечение должно удовлетворять следующим требованиям:

— программа получает на вход размер очереди;
— программа выдает журналирвоание работы конвейера;
— генератор строк выдает ненулевые строки, состоящие из строчных и заглавных букв;
— шифрование каждым методом выполняется с разными ключами.

3.2 Средства реализации

Для реализации ПО был выбран компилируемый многопоточный язык программирования Golang, поскольку язык отличается легкой и быстрой сборкой программ и автоматическим управлением памяти.

3.3 Листинг программы

В приведенных ниже листингах представлены следующие реализации:

1) паралельного конвейера (листинги 3.1, 3.2, 3.3);
2) синхронного конвейера (листинги 3.4, 3.5);
3) XOR-шифра (листинг 3.6);
4) шифра Цезаря (листинг 3.7).

Листинг 3.1 — Реализация параллельного конвейера

```
1 func Pipeline(count int, ch chan int) *Queue {
2
3     first := make(chan *PipeTask, count)
4     second := make(chan *PipeTask, count)
5     third := make(chan *PipeTask, count)
6
7     line := initQueue(count)
```

Листинг 3.2 — Реализация параллельного конвейера

```
    gen_string := func() {
        for {
            select {
            case pipe_task := <-first:
                pipe_task.generated = true

                len_s := rand.Intn(100)
                pipe_task.source = chiperalg.GenerateRune(len_s)
                pipe_task.start_generating = time.Now()
                pipe_task.dst = chiperalg.XorChiperFirst(pipe_task.source, len_s)

                pipe_task.end_generatig = time.Now()

                second <- pipe_task
            }
        }
    }

    get_table := func() {
        for {
            select {
            case pipe_task := <-second:
                pipe_task.skip_table_made = true

                pipe_task.start_table = time.Now()

                pipe_task.dst = chiperalg.CaesarChiper(pipe_task.dst,
                    len(pipe_task.dst))
                pipe_task.end_table = time.Now()

                third <- pipe_task
            }
        }
    }
    match := func() {
        for {
            select {
            case pipe_task := <-third:
                pipe_task.pattern_mached = true

                pipe_task.start_match = time.Now()
                pipe_task.dst = chiperalg.XorChiperSecond(pipe_task.dst,
                    len(pipe_task.dst))
                pipe_task.end_match = time.Now()
```

Листинг 3.3 — Реализация параллельного конвейера

```
1
2                   line.enqueue(pipe_task)
3                   if pipe_task.num == count-1 {
4                       ch <- 0
5                   }
6               }
7           }
8       }
9
10      go gen_string()
11      go get_table()
12      go match()
13
14      for i := 0; i < count; i++ {
15          pipe_task := new(PipeTask)
16          pipe_task.num = i + 1
17          first <- pipe_task
18      }
19      return line
20  }
```

Листинг 3.4 — Реализация синхронного конвейера

```
1  func chiper_xor_first(task *PipeTask) *PipeTask {
2      task.generated = true
3
4      len_s := rand.Intn(100)
5      task.source = chiperalg.GenerateRune(len_s)
6      task.start_generating = time.Now()
7      task.dst = chiperalg.XorChiperFirst(task.source, len_s)
8
9      task.end_generatig = time.Now()
10
11     return task
12 }
13
14 func chiper_caesar(task *PipeTask) *PipeTask {
15     task.skip_table_made = true
16
17     task.start_table = time.Now()
```

Листинг 3.5 — Реализация синхронного конвейера

```
    task.dst = chiperalg.CaesarChiper(task.dst, len(task.dst))
    task.end_table = time.Now()

    return task
}

func chiper_xor_second(task *PipeTask) *PipeTask {
    task.pattern_mached = true

    task.start_match = time.Now()
    task.dst = chiperalg.XorChiperSecond(task.dst, len(task.dst))
    task.end_match = time.Now()

    return task
}

func Sync(count int) *Queue {

    line_first := initQueue(count)
    line_second := initQueue(count)
    line_third := initQueue(count)

    for i := 0; i < count; i++ {
        pipe_task := new(PipeTask)
        pipe_task = chiper_xor_first(pipe_task)
        line_first.enqueue(pipe_task)
        if len(line_first.queue) != 0 {
            pipe_task = chiper_caesar(line_first.dequeue())
            line_second.enqueue(pipe_task)
            if len(line_second.queue) != 0 {
                pipe_task = chiper_xor_second(line_second.dequeue())
                line_third.enqueue(pipe_task)
            }
        }
    }
    return line_third
}
```

Листинг 3.6 — Реализация алгоритма XOR шифрования

```
func XorChiperFirst(src []rune, size int) []rune {

    var smeshenie = 8
    var itog = make([]rune, size)
    for i := range src {
        itog[i] = rune(int(src[i]) ^ smeshenie)
    }
    return itog
}

func XorChiperSecond(src []rune, size int) []rune {

    var smeshenie = 3
    var itog = make([]rune, size)
    for i := range src {
        itog[i] = rune(int(src[i]) ^ smeshenie)
    }
    return itog
}
```

Листинг 3.7 — Реализация алгоритма шифра Цезаря

```
func CaesarChiper(src []rune, size int) []rune {
    var letters = []rune("abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ")
    var smeshenie = 6
    var itog = make([]rune, size)
    for i := range src {
        var mesto = search(letters, src[i])
        var newmesto = (mesto + smeshenie) % len(letters)
        if search(letters, src[i]) > -1 {
            itog[i] = letters[newmesto]
        } else {
            itog[i] = src[i]
        }
    }
    return itog
}
```

3.4 Тестирование ПО

Тестирование ПО проводилось написанием журналирования для вывода содержания заявок очереди

Листинг 3.8 — Тестирование ПО

```
func Log(qu *Queue) {
    fmt.Println("")
    line := qu.queue
    for i := range line {
        if line[i] != nil {
            fmt.Printf("%3v & %10v & %10v & %10v & %10v &  \n", i,
                string(line[i].source), string(line[i].dst1), string(line[i].dst2),
                string(line[i].dst3))
        }
    }
}
```

В таблице 3.1 отображёны результаты на очереди размером в 5 элементов.

Таблица 3.1 — Тесты проверки корректности программы

№	Source	XOR key-1	Caesar	XOR key-2
1	k	c	i	j
2	sBtgPSb	{J|oX[j	{P|ud[p	xS_vgXs
3	cw	k_	q_	r|
4	XDpVR	PLx^Z	VRD^f	UQG]e
5	LMwwVtm	DE__^|e	JK__^|k	IH||]_h

Вывод

Было написано и протестировано программное обеспечение для решения поставленной задачи.

4 Экспериментальный раздел

Раздел содержит результат сравнительного анализа работы конвейера, работающего на одном потоке и многопоточного конвейера, приведена таблица результатов эксперимента и дана оценка эффективности предложенной в работе реализации алгоритма.

4.1 Технические характеристики

Тестирование выполнялось на устройстве со следующими техническими характеристиками:

— операционная система: macOS Ventura 13.0 [5];
— оперативная память: 8 Gb;
— процессор: Apple M1 [6].

4.2 Описание системы тестирования

Получение характеристик работы очереди осуществлялось с помощью установки штампов на начало и завершение работы каждой из лент конвейера. Работа с временными штампами продемонстрирована на листинге 3.2

4.3 Постановка эксперимента

В данном эксперименте тестируется влияние распараллеливания конвейерной обработки на время работы конвейера. Эксперимент проведен на данных типа rune, который в языке Golang является псевдонимом для целочисленного типа размерностью 32 бита [7]. Было выполнено одно снятие временных штампов. Данные не усреднялись. Во время тестирования устройство было подключено к блоку питания и не нагружено никакими приложениями, кроме встроенных приложений окружения, окружением и системой тестирования. Оптимизация компилятора была отключена.

4.4 Результаты эксперимента

Результаты эксперимента представлены в таблице 4.1.

Из таблицы можно сделать вывод, что распараллеленный конвейер выполняет работу на 6% быстрее, чем синхронный. Таблица с результатами анализа приведена ниже(4.2):

При распараллеливании конвейера на три потока возникает ситуация, когда простоя в очереди нет вовсе. Так же время простоя первой, второй и третьей линии больше на синхронном конвейере в 2.8, 1,2 и 1,7 раз, чем на параллельном соответственно.

Однако, среднее время заявки в системе на синхронном конвейере в ≈ 2.3 раза меньше, чем время заявки в синхронном конвейере. Соответственно, выигрыш происходит исключительно за счет обеспечения меньшего простоя очереди и ситуаций, когда простоя нет вовсе.

Таблица 4.1 — Замеры времени работы на очереди размером 20

№	Начало обработки заявки					
	Параллельно, ns			Синхронно, ns.		
	Линия 1	Линия 2	Линия 3	Линия 1	Линия 2	Линия 3
1	0	53934	111522	0	40521	41301
2	42507	96550	112526	41805	83047	83872
3	87564	240258	248300	84239	124716	126116
4	133080	244788	249092	126667	167664	168461
5	174362	246233	249644	168860	209765	210664
6	214072	318217	341519	211164	256655	274355
7	261739	320344	342130	275205	323431	324521
8	302760	472094	482306	325112	366840	368582
9	348170	477270	483114	369605	413718	414936
10	389616	479263	483587	415422	463733	465791
11	429963	480493	484016	466713	507510	508254
12	470433	593634	600897	508604	548992	550028
13	516046	597457	601563	550442	590621	591488
14	556898	598806	601930	591959	632602	633636
15	596784	716283	743636	634021	674088	677474
16	641117	719814	744380	677841	718884	720048
17	680814	741599	744850	735219	777608	779706
18	722195	850056	876476	780280	852850	853711
19	767730	873968	877044	854184	894956	896102
20	808537	875006	878601	896482	936600	937634

Таблица 4.2 — Анализ временных замеров

Характеристика		Параллельно, ns,			Синхронно, ns.		
Линия		1	2	3	1	2	3
Простой очереди	gen.	23807	700566	761270	68686	858165	891498
	min	0	0	0	1200	40554	41244
	max	5421	141752	141500	18744	73787	73762
	avg	1190	35028	38063	3434	42908	44574
Время заявки в системе	min	45238			41325		
	max	179982			73655		
	avg	102424			43826		

4.5 Вывод

Конвейер, реализованный на параллельных процессах обеспечивает ситуации с минимальным или вовсе отсутствующим простоем лент. Однако, за счет затрат на обеспечение реентерабельности функций, работы с атомарными операциями и буферизации потоков, в среднем заявка находится в системе дольше, чем заявка в синхронном конвейере.

Заключение

В рамках данной лабораторной работы была достигнута её цель: изучены конвеерные вычисления. Также выполнены следующие задачи:

— была изучена организация конвеерной обработки данных;
— были изучены и реализованы шифр Цезаря и xor-шифр;
— были проведены замеры процессорного времени работы для реализованных алгоритмов.

Вычислительный конвейер был реализован на трех потоках и на одном потоке. Алгоритмы шифрования Цезаря и XOR-шифра были применены для обработки конвейерными линиями. Эксперимент показал, что распараллеливание вычислительного конвейера приводит к выигрышу в 6%. Выигрыш происходит исключительно за счет обеспечения меньшего простоя очереди и ситуаций, когда простоя нет вовсе.

Список источников

1) Pipelining: Basic Concepts and Approaches // [Электронный ресурс]. Режим доступа: https://www.ijser.org/researchpaper/Pipelining-Basic-Concepts-And-Approaches.pdf дата обращения: 10.12.2022);

2) Шифр Цезаря или как просто зашифровать текст. // [Электронный ресурс]. Режим доступа: https://habr.com/ru/post/534058/ (дата обращения: 10.12.2022);

3) Шифрование методом XOR. // [Электронный ресурс]. Режим доступа: https://evileg.com/ru/post/271/ (дата обращения: 10.12.2022).

4) Donald Knuth. “The Art of Computer Programming. Volume 1: Fundamental Algorithms, Third Edition.” в: Addison-Wesley, 1997. гл. Stacks, Queues, and Dequeues, с. 238—243.

5) Документация по операционной системе macOS Ventura. [Электронный ресурс]. Режим доступа: https://www.apple.com/macos/ventura/ (дата обращения: 10.12.2022).

6) Документация по процессору Apple M1. [Электронный ресурс]. Режим доступа: https://www.apple.com/ru/newsroom/2020/11/apple-unleashes-m1/ (дата обращения: 10.12.2022).

7) Rune in Golang. [Электронный ресурс]. Режим доступа: https://www.geeksforgeeks.org/rune-in-golang/ (дата обращения: 10.12.2022).