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 №5 по курсу "Анализ алгоритмов"

Тема Алгоритм конвейерной обработки

Студент Богаченко А.Е.

Группа ИУ7-56Б

Оценка (баллы) _____

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

Москва — 2020 г.

Оглавление

Введение

Цель работы: изучение возможности конвеерной обработки и использование такого подхода на практике. Необходимо сравнить времени работы алгоритма на нескольких потоков и линейную реализацию.

В ходе лабораторной работы предстоит:

- реализовать конвейер на потоках;
- реализовать линейную обработку;
- провести сравнение времени работы.

1 Аналитическая часть

Конвейер - система поточного производства [1]. В терминах программирования ленты конвейера представлены функциями, выполняющими над неким набором данных операции и предающие их на следующую ленту конвейера. Моделирование конвейерной обработки хорошо сочетается с технологией многопоточного программирования - под каждую ленту конвейера выделяется отдельный поток, все потоки работают в асинхронном режиме.

В качестве предметной области было выбрано создание личной карточки пользователя - на первой линии конвейера происходит создание имени, на второй создание почтового адреса, на третьей создание любимого напитка.

1.1 Параллельное программирование

При использовании многопроцессорных вычислительных систем с общей памятью [2] обычно предполагается, что имеющиеся в составе системы процессоры обладают равной производительностью, являются равноправными при доступе к общей памяти, и время доступа к памяти является одинаковым (при одновременном доступе нескольких процессоров к одному и тому же элементу памяти очерёдность и синхронизация доступа обеспечивается на аппаратном уровне). Многопроцессорные системы подобного типа, обычно, именуются симметричными мультипроцессорами (`symmetric multiprocessors, SMP`) [3].

Перечисленному выше набору предположений удовлетворяют также активно развиваемые в последнее время многоядерные процессоры [4], в которых каждое ядро представляет практически независимо функционирующее вычислительное устройство.

Обычный подход при организации вычислений для многопроцессорных вычислительных систем с общей памятью – создание новых параллельных методов на основе обычных последовательных программ, в которых или автоматически компилятором, или непосредственно программистом выделяются участки независимых друг от друга вычислений. Возможности автоматического анализа программ для порождения параллельных вычисле-

ний достаточно ограничены [5], и второй подход является преобладающим. При этом для разработки параллельных программ могут применяться как новые «новые» языки [6], поддерживающие параллельное программирование, так и уже имеющиеся языки [7], расширенные некоторым набором операторов для параллельных вычислений. Также стоит обратить внимание на готовые пакеты для разработки, сочетающие в себе различные инструменты профилирования, компиляторы и анализаторы [8].

Параллельное программирование реализуемо либо посредством использования библиотек, обеспечивающих определённый программный интерфейс (`API`) для разработки параллельных программ [9] либо посредством встроенных в язык интерфейсов [10], если таковые имеются.

Вывод

Была рассмотрена конвейерная обработка данных, технология параллельного программирования и организация многопроцессорных вычислительных систем.

2 Конструкторская часть

Требования к вводу:

На ввод подается целое число - желаемое колличество изготовленных экземпляров

Требования к программе:

- вывод статистики обаботанных экземпляров.

2.1 Распараллеливание программы

Распараллеливание программы должно ускорять время работы. Это достигается за счет перенесения каждой из лент конвейера на отдельный поток.

Вывод

В данном разделе былы рассмотрены способы распараллеливания конвейера.

3 Технологическая часть

В данном разделе приведены требования к программному обеспечению, средства реализации и листинги кода.

3.1 Требования к ПО

К программе предъявляется ряд требований:

- корректная сортировка.

3.2 Структура ПО

В данном разделе будет рассмотрена структура ПО 3.1.

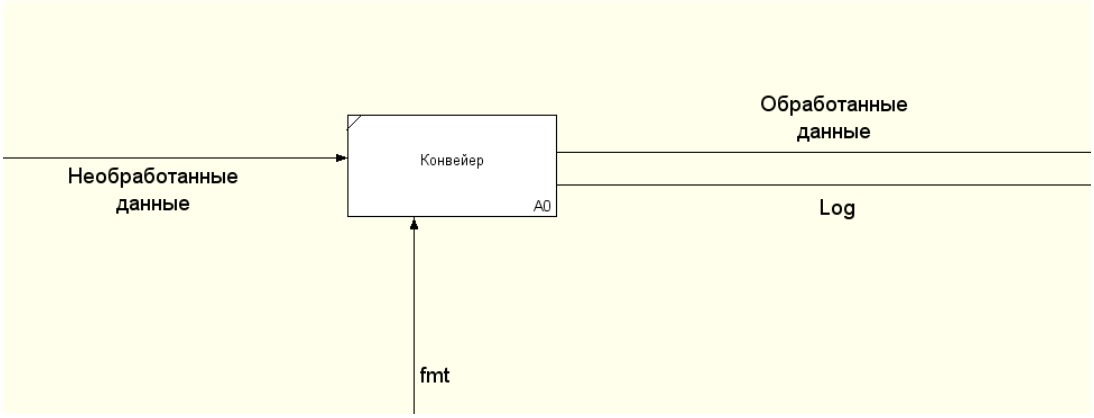

Рисунок 3.1 – Структура ПО

3.3 Средства реализации

В качестве языка программирования для реализации данной лабораторной работы был выбран многопоточный язык GO [7]. Данный выбор обусловлен моим желанием расширить свои знания в области применения данного язкыа. Так же данный язык предоставляет средства тестирования разработанного ПО.

Время работы алгоритмов было замерено с помощью функции `Now()` из библиотеки `Time` [11].

3.4 Листинг кода

В листингах 3.1 – 3.2 приведены реализации параллельного и линейного конвейеров.

Листинг 3.1 – Реализация параллельного конвейера

```
func Parallel(amount int, wait chan int) *Queue {
    f := make(chan *Person, 5)
    s := make(chan *Person, 5)
    t := make(chan *Person, 5)
    line := createQueue(amount)
    name := func() {
        for {
            select {
            case p := <-f:
                p.haveName = true

                p.startName = time.Now()
                randomOps()
                p.name = gofakeit.Name()
                randomOps()
                p.doneName = time.Now()

                s <- p
            }
        }
    }

    email := func() {
        for {
            select {
            case p := <-s:
                p.haveEmail = true

                p.startEmail = time.Now()
                randomOps()
                p.email = gofakeit.Email()
                randomOps()
                p.doneEmail = time.Now()

                t <- p
            }
        }
    }

    beer := func() {
```

```
        for {
            select {
            case p := <-t:
                p.haveBeer = true

                p.startBeer = time.Now()
                randomOps()
                p.beer = gofakeit.BeerName()
                randomOps()
                p.doneBeer = time.Now()

                line.push(p)
                if p.num == amount {
                    wait <- 0
                }

            }
        }
    }

    go name()
    go email()
    go beer()
    for i := 0; i <= amount; i++ {
        p := new(Person)
        p.num = i
        f <- p
    }

    return line
}
```

Листинг 3.2 – Реализация линейного конвейера

```
func name(p *Person, qEmail *Queue) {
    p.haveName = true

    p.startName = time.Now()
    randomOps()
    p.name = gofakeit.Name()
    randomOps()
    p.doneName = time.Now()

    qEmail.push(p)
}

func email(p *Person, qBeer *Queue) {
    p.haveEmail = true

```

```
16      p.startEmail = time.Now()
17      randomOps()
18      p.email = gofakeit.Email()
19      randomOps()
20      p.doneEmail = time.Now()
21
22      qBeer.push(p)
23  }
24
25  func beer(p *Person, finished *Queue) {
26      p.haveBeer = true
27
28      p.startBeer = time.Now()
29      randomOps()
30      p.beer = gofakeit.BeerName()
31      randomOps()
32      p.doneBeer = time.Now()
33
34      finished.push(p)
35  }
36
37  func Linear(amount int) *Queue {
38      qEmail := createQueue(amount)
39      qBeer := createQueue(amount)
40      finished := createQueue(amount)
41      i := 0
42      for i != -1 {
43          p := new(Person)
44          p.num = i
45          name(p, qEmail)
46          if qEmail.l >= 0 {
47              email(qEmail.pop(), qBeer)
48          }
49          if qBeer.l >= 0 {
50              beer(qBeer.pop(), finished)
51          }
52          if finished.q[len(finished.q)-1] != nil {
53              return finished
54          }
55          i++
56      }
57      return finished
58  }
```

Вывод

В данном разделе была рассмотрена структура ПО и листинги кода программы.

4 Исследовательская часть

4.1 Пример работы

Лог линейной обработки представлен на рисунке 4.1, лог конвейерной обработки представлен на рисунке 4.2

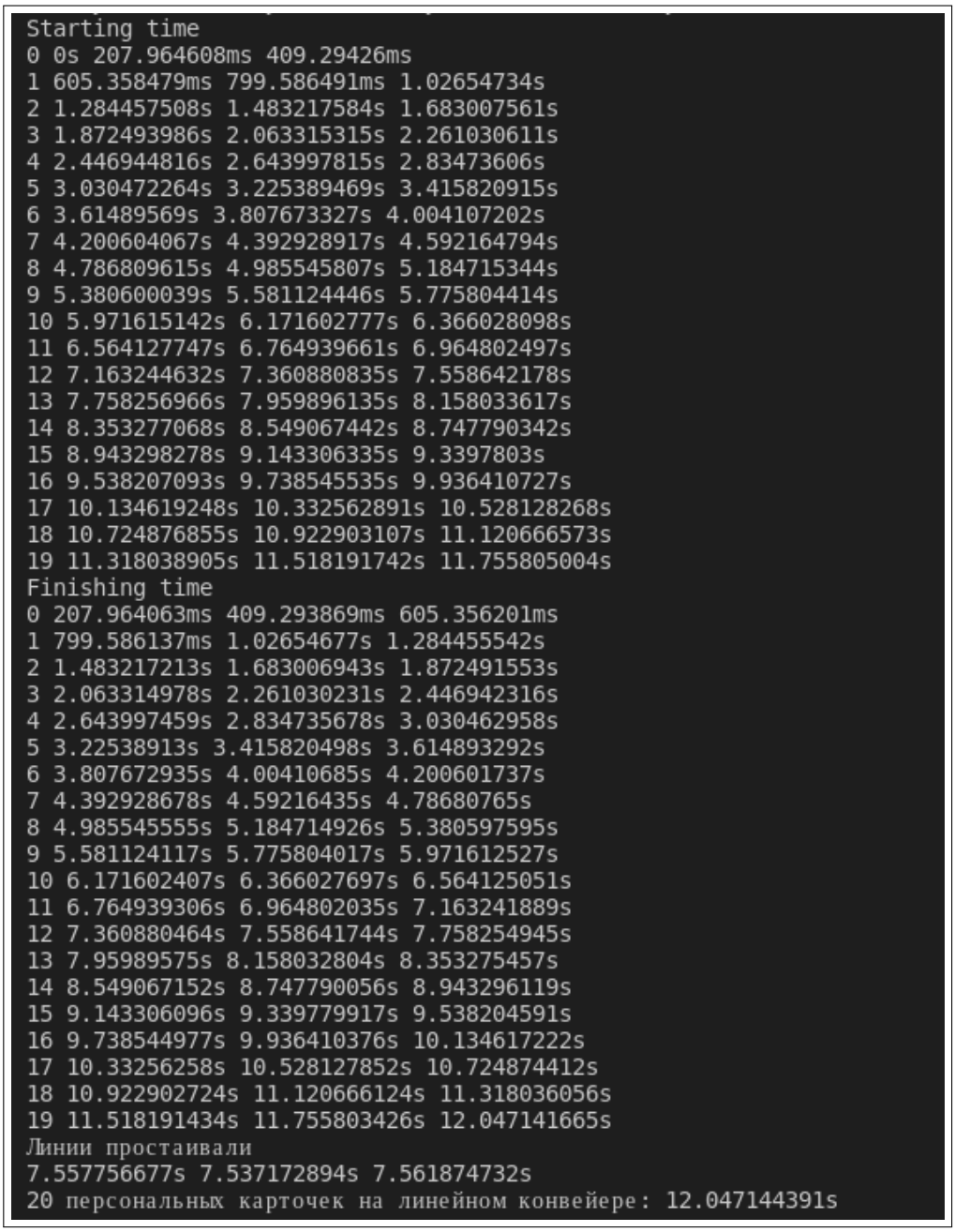

Рисунок 4.1 – Лог работы линейной обработки

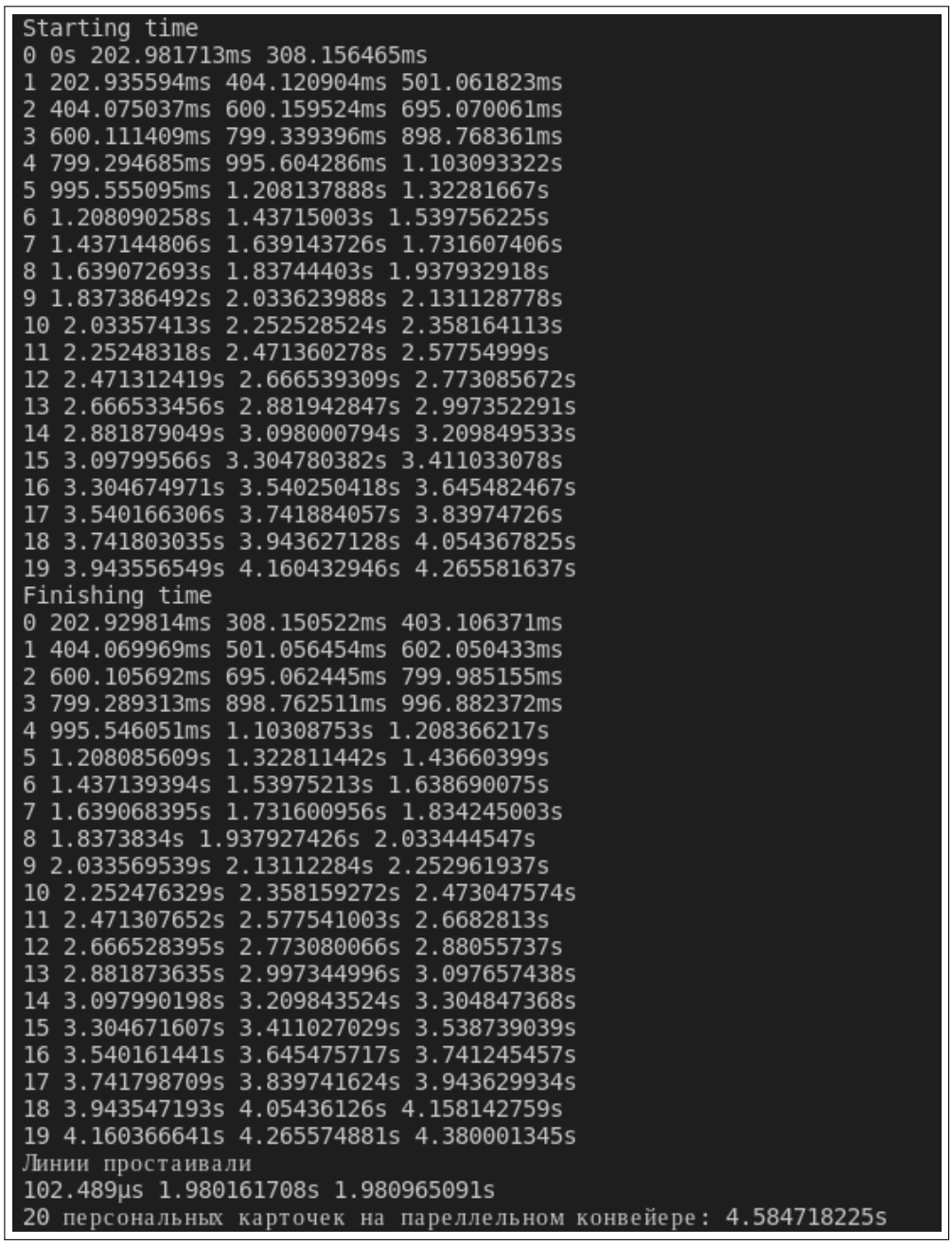

Рисунок 4.2 – Лог работы конвейерной обработки

4.2 Технические характеристики

Технические характеристики устройства, на котором выполнялось тестирование:

- Операционная система: Kali [12] Linux [13] 5.9.0-kali1-amd64.
- Память: 8 GB.

- Процессор: Intel® Core™ i5-8250U [14] CPU @ 1.60GHz
- Количество логических потоков: 8

Тестирование проводилось на ноутбуке при включённом режиме производительности. Во время тестирования ноутбук был нагружен только системными процессами.

4.3 Время выполнения алгоритмов

Алгоритмы тестировались при помощи написания «бенчмарков» [15], предоставляемых встроенными в Go средствами. Для такого рода тестирования не нужно самостоятельно указывать количество повторов. В библиотеке для тестирования существует константа N, которая динамически варьируется в зависимости от того, был ли получен стабильный результат или нет.

В листинге 4.1 пример реализации бенчмарка.

Листинг 4.1 – Реализация бенчмарка

```
func BenchmarkParallel10(b *testing.B) {
    for i := 0; i < b.N; i++ {
        wait := make(chan int)
        Parallel(10, wait)
        <-wait
    }
}
```

На рисунках 4.3 приведён график сравнения производительности конвейеров.

4.4 Производительность алгоритмов

Производительность и объем выделенной памяти при работе алгоритмов указаны на рисунке 4.4.

Вывод

Экперимент показывает, что использование нескольких потоков для реализации конвейерной обработки данных ускоряет алгоритм в несколько раз. При этом возникает ситуация при которой ленты не простаивают. Тратится лишь малое время для передачи данных на линию.

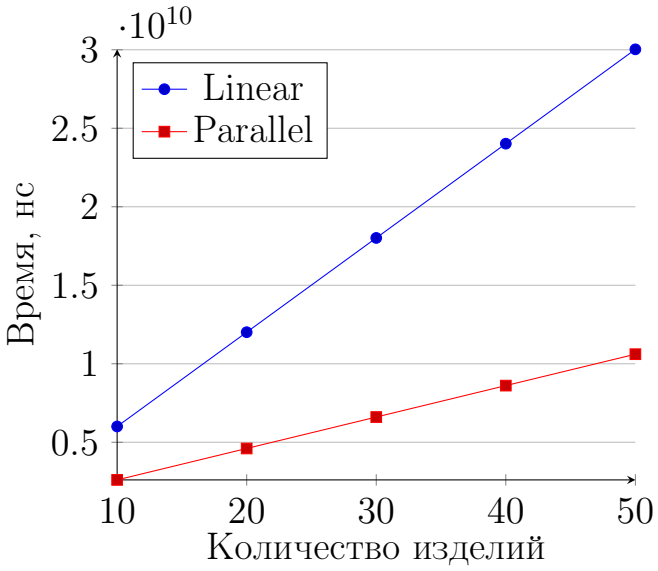

Рисунок 4.3 – Сравнение конвейеров.

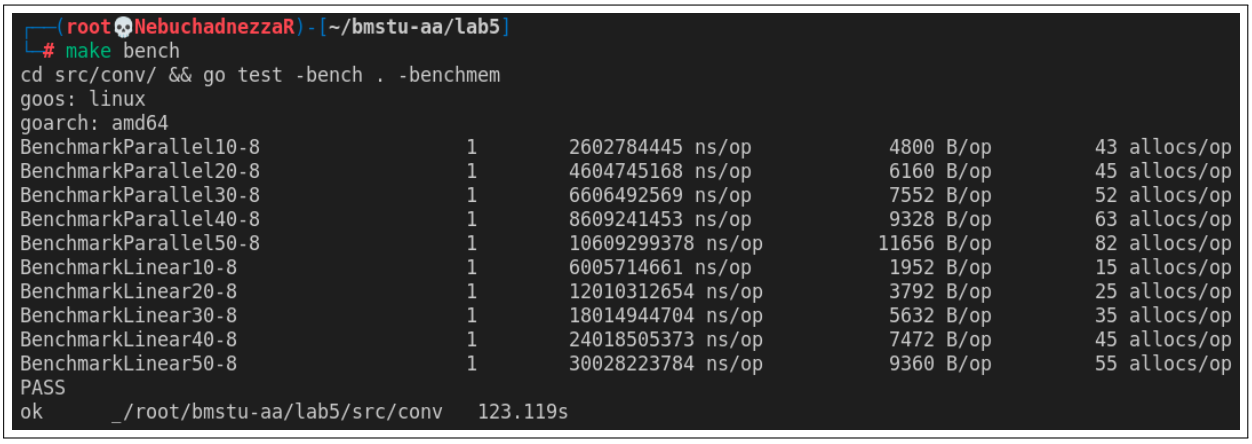

Рисунок 4.4 – Замеры производительности алгоритмов, выполненные при помощи команды `go test -bench . -benchmem`

Заключение

В ходе лабораторной работы были изучены возможности параллельных вычислений, были реализованы алгоритмы последовательной и параллельной конвейерной обработки данных.

Было проведено тестирование разработанных алгоритмов, выполнены замеры времени и проведён сравнительный анализ временной эффективности алгоритмов.

Экспериментально были установлены преимущества использования параллельного конвейера.

Литература

[1] В. П. Меднов Е. П. Бондарев. Транспортные, распределительные и рабочие конвейеры. – М, 1970.

[2] Гергель В.П. Стронгин Р.Г. Основы параллельных вычислений для многопроцессорных вычислительных систем. – Н.Новгород, ННГУ, 2003.

[3] SMP Sarah L. Harris, David Money Harris, in Digital Design and Computer Architecture, 2016 [Электронный ресурс]. Режим доступа: `https://www.sciencedirect.com/topics/computer-science/symmetric-multi-processor` (дата обращения: 18.12.2020).

[4] Центральный процессор «Эльбрус-4С» [Электронный ресурс]. Режим доступа: `http://www.mcst.ru/mikroprocessor-elbrus4s` (дата обращения: 18.12.2020).

[5] Padua David. Parallelization, Automatic // Encyclopedia of Parallel Computing. Springer US, 2011. С. 1442–1450.

[6] Асинхронное программирование в F♯ [Электронный ресурс]. Режим доступа: `https://docs.microsoft.com/ru-ru/dotnet/fsharp/tutorials/asynchronous-and-concurrent-programming/async` (дата обращения: 18.12.2020).

[7] The Go Programming Language [Электронный ресурс]. Режим доступа: `https://golang.org/` (дата обращения: 09.09.2020).

[8] Intel Parallel Studio XE [Электронный ресурс]. Режим доступа: `https://software.intel.com/content/www/us/en/develop/tools/oneapi/commercial-base-hpc.html` (дата обращения: 18.12.2020).

[9] Thread library [Электронный ресурс]. Режим доступа: `https://en.cppreference.com/w/cpp/thread` (дата обращения: 18.12.2020).

[10] testing – GO channels <- concurrency <- [Электронный ресурс]. Режим доступа: `https://tour.golang.org/concurrency/2` (дата обращения: 18.12.2020).

[11] Package time [Электронный ресурс]. Режим доступа: https://golang.org/pkg/time/ (дата обращения: 18.11.2020).

[12] Our Most Advanced Penetration Testing Distribution, Ever. [Электронный ресурс]. Режим доступа: https://kali.org/ (дата обращения: 12.09.2020).

[13] Linux – Википедия [Электронный ресурс]. Режим доступа: https://ru.wikipedia.org/wiki/Linux (дата обращения: 12.09.2020).

[14] Intel Processors [Электронный ресурс]. Режим доступа: https://www.intel.com/content/www/us/en/products/processors/core/i5-processors.html (дата обращения: 12.09.2020).

[15] Testing – The Go Programming Language [Электронный ресурс]. Режим доступа: https://golang.org/pkg/testing/ (дата обращения: 12.09.2020).