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 Конвейерные вычисления

Студент Кононенко С.С.

Группа ИУ7-53Б

Оценка (баллы) _____

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

Москва — 2020 г.

Оглавление

Введение

Целью данной лабораторной работы является изучение конвейерных вычислений.

При обработке данных могут возникать ситуации, когда один набор данных необходимо обработать последовательно несколькими алгоритмами. В таком случае удобно использовать конвейерную обработку данных, что позволяет на каждой следующей «линии» конвейера использовать данные, полученные с предыдушего этапа.

Отдельно стоит упомянуть асинхронные конвейерные вычисления. Отличие от линейных состоит в том, что при таком подходе линии работают с меньшим временем простоя, так как могут обрабатывать задачи независимо от других линий.

Задачи лабораторной работы:

- рассмотреть и изучить асинхронную конвейерную обработку данных;
- реализовать систему конвейерных вычислений с количеством линий не меньше трех;
- на основании проделанной работы сделать выводы.

1 Аналитическая часть

В данном разделе представлены теоретические сведения о рассматриваемых алгоритмах.

1.1 Конвейерная обработка данных

Конвейер — способ организации вычислений, используемый в современных процессорах и контроллерах с целью повышения их производительности (увеличения числа инструкций, выполняемых в единицу времени — эксплуатация параллелизма на уровне инструкций), технология, используемая при разработке компьютеров и других цифровых электронных устройств.

Идея заключается в параллельном выполнении нескольких инструкций процессора. Сложные инструкции процессора представляются в виде последовательности более простых стадий. Вместо выполнения инструкций последовательно (ожидания завершения конца одной инструкции и перехода к следующей), следующая инструкция может выполняться через несколько стадий выполнения первой инструкции. Это позволяет управляющим цепям процессора получать инструкции со скоростью самой медленной стадии обработки, однако при этом намного быстрее, чем при выполнении эксклюзивной полной обработки каждой инструкции от начала до конца.

Многие современные процессоры управляются тактовым генератором. Процессор внутри состоит из логических элементов и ячеек памяти — триггеров. Когда приходит сигнал от тактового генератора, триггеры приобретают своё новое значение, и «логике» требуется некоторое время для декодирования новых значений. Затем приходит следующий сигнал от тактового генератора, триггеры принимают новые значения, и так далее. Разбивая последовательности логических элементов на более короткие и помещая триггеры между этими короткими последовательностями, уменьшают время, необходимое логике для обработки сигналов. В этом случае длительность одного такта процессора может быть соответственно уменьшена.

1.2 Предметная область лабораторной работы

В качестве алгоритма, реализованного для распределения на конвейере, было выбрано абстрактное оформление кредитной карты в банке, состоящее из трех этапов:

- генерация номера карты;
- генерация CVV карты;
- генерация даты валидности карты.

Вывод

В данной работе стоит задача реализации асинхронных конвейерных вычислений. Были рассмотрены особенности построения конвейерных вычислений.

2 Конструкторская часть

В данном разделе представлены схемы рассматриваемых алгоритмов.

2.1 Разработка алгоритмов

На рисунке 2.1 приведена схема организации конвейерных вычислений.

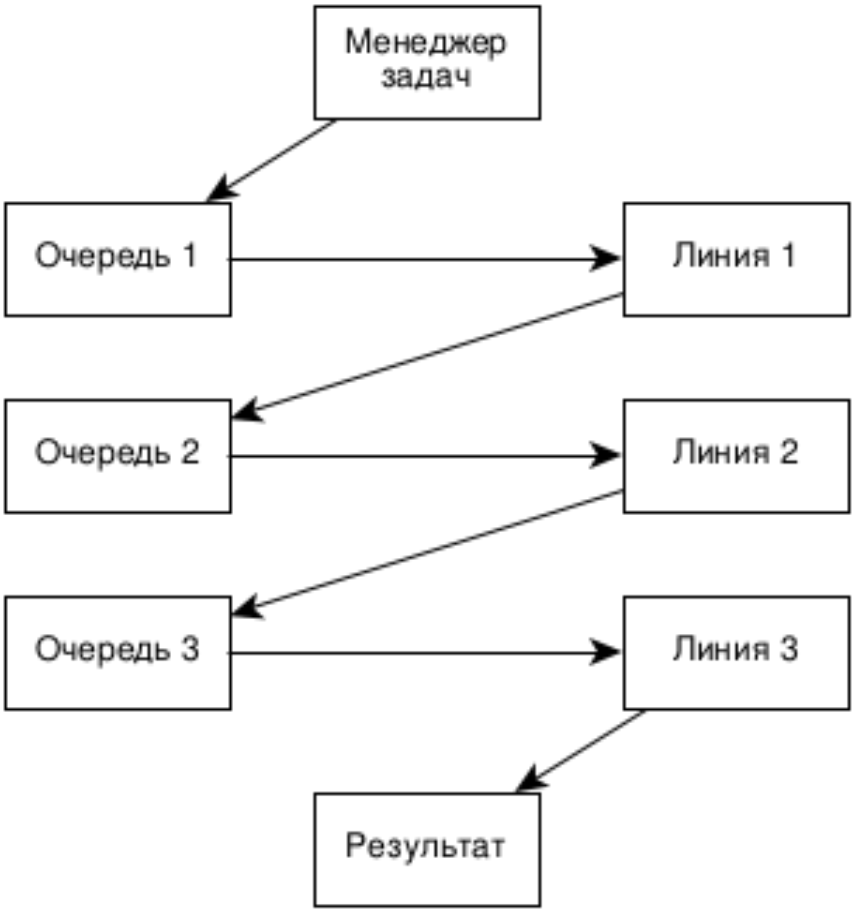

Рисунок 2.1 – Схема организации конвейерных вычислений

Вывод

На основе теоретических данных, полученных из аналитического раздела, были построены схемы требуемых алгоритмов.

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:

- на вход подаётся количество задач (заявок на оформление кредитной карты);
- на выходе — время, затраченное на обработку заявок;
- в процессе обработки задач необходимо фиксировать время прихода и ухода заявки с линии.

3.2 Средства реализации

В качестве языка программирования для реализации данной лабораторной работы был выбран многопоточный язык Golang [1]. Данный выбор обусловлен моим желанием расширить свои знания в области применения данного язкыа а также встроенным в язык средством организации параллельных вычислений. Помимо этого, встроенные средства языка предоставляют высокоточные средства тестирования разработанного ПО.

3.3 Листинг кода

В листингах 3.1, 3.2, 3.3 и 3.4 приведены реализации конвейерных вычислений, дополнительные типы данных, утилиты для работы с очередями и инструмент логгирования соответственно.

Листинг 3.1 – Реализация конвейерных вычислений

```
package conv

import (
    "time"

    "github.com/brianvoe/gofakeit"
)

// Conveyor used to generate credit cards in parallel conveyor way.
func Conveyor(n int, ch chan int) *Queue {
    f := make(chan *CreditCard, 5)
    s := make(chan *CreditCard, 5)
    t := make(chan *CreditCard, 5)
    l := createQueue(n)

    number := func() {
        for {
            select {
            case cc := <-f:
                cc.haveNumber = true

                cc.startNumber = time.Now()
                cc.number = gofakeit.CreditCardNumber(nil)
                cc.doneNumber = time.Now()

                s <- cc
            }
        }
    }

    cvv := func() {
        for {
            select {
            case cc := <-s:
                cc.haveCvv = true

                cc.startCvv = time.Now()
                cc.cvv = gofakeit.CreditCardCvv()
                cc.doneCvv = time.Now()

                t <- cc
            }
        }
    }

    exp := func() {
        for {
```

```
48              select {
49              case cc := <-t:
50                  cc.haveExp = true
51
52                  cc.startExp = time.Now()
53                  cc.exp = gofakeit.CreditCardExp()
54                  cc.doneExp = time.Now()
55
56                  l.push(cc)
57                  if cc.num == n {
58                      ch <- 0
59                  }
60
61              }
62          }
63      }
64
65      go number()
66      go cvv()
67      go exp()
68      for i := 0; i <= n; i++ {
69          cc := new(CreditCard)
70          cc.num = i
71          f <- cc
72      }
73
74      return l
75  }
76
77  func number(cc *CreditCard, qNumber *Queue) {
78      cc.haveNumber = true
79
80      cc.startNumber = time.Now()
81      cc.number = gofakeit.CreditCardNumber(nil)
82      cc.doneNumber = time.Now()
83
84      qNumber.push(cc)
85  }
86
87  func cvv(cc *CreditCard, qCvv *Queue) {
88      cc.haveCvv = true
89
90      cc.startCvv = time.Now()
91      cc.cvv = gofakeit.CreditCardCvv()
92      cc.doneCvv = time.Now()
93
94      qCvv.push(cc)
95  }
```

```
96
97  func exp(cc *CreditCard, done *Queue) {
98      cc.haveExp = true
99
100     cc.startExp = time.Now()
101     cc.exp = gofakeit.CreditCardExp()
102     cc.doneExp = time.Now()
103
104     done.push(cc)
105 }
```

Листинг 3.2 – Реализация дополнительных типов данных

```
1  package conv
2
3  import "time"
4
5  type CreditCard struct {
6      num int
7      haveNumber bool
8      haveCvv bool
9      haveExp bool
10     startNumber time.Time
11     doneNumber time.Time
12     startCvv time.Time
13     doneCvv time.Time
14     startExp time.Time
15     doneExp time.Time
16
17     number string
18     cvv string
19     exp string
20 }
21
22 type Queue struct {
23     q []*CreditCard
24     l int
25 }
```

Листинг 3.3 – Реализация утилит работы с очередью

```
1  package conv
2
3  func createQueue(amount int) *Queue {
4      q := new(Queue)
5      q.q = make([]*CreditCard), amount, amount)
6      q.l = -1
7
8      return q
```

```
}

func (q *Queue) push(p *CreditCard) {
    if q.l != len(q.q)-1 {
        q.q[q.l+1] = p
        q.l++
    }
}

func (q *Queue) pop() *CreditCard {
    p := q.q[0]
    q.q = q.q[1:]
    q.l--

    return p
}
```

Листинг 3.4 – Реализация инструмента логгирования

```
package conv

import (
    "fmt"
    "strings"
    "time"

    "github.com/logrusorgru/aurora"
)

// Log used to log conveyor tasks time.
func Log(q *Queue, logCreditCard bool) {
    var (
        fdtime time.Duration
        sdtime time.Duration
        tdtime time.Duration
    )

    l := q.q
    start := l[0].startNumber

    fmt.Printf("%38v\n", aurora.BgRed("STARTING␣TIME"))
    fmt.Printf("%v%36v%v\n", "+", strings.Repeat("-", 36), "+")
    fmt.Printf(
        "|%3v|%10v|%10v|%10v|\n",
        aurora.Green("N"), aurora.Green("CardNumber"), aurora.Green("CardCVV"),
            aurora.Green("CardExp"),
    )
    fmt.Printf("%v%36v%v\n", "+", strings.Repeat("-", 36), "+")
    for i := 0; i < len(l); i++ {
```

```
30          if l[i] != nil {
31              fmt.Printf(
32                  "|%3v|%10v|%10v|%10v|\n",
33                  i, l[i].startNumber.Sub(start), l[i].startCvv.Sub(start),
                        l[i].startExp.Sub(start),
34              )
35          }
36      }
37      fmt.Printf("%v%36v%v\n", "+", strings.Repeat("-", 36), "+")
38
39      fmt.Printf("%38v\n", aurora.BgRed("FINISHING␣TIME"))
40      fmt.Printf("%v%36v%v\n", "+", strings.Repeat("-", 36), "+")
41      fmt.Printf(
42          "|%3v|%10v|%10v|%10v|\n",
43          aurora.Green("N"), aurora.Green("CardNumber"), aurora.Green("CardCVV"),
                aurora.Green("CardExp"),
44      )
45      fmt.Printf("%v%36v%v\n", "+", strings.Repeat("-", 36), "+")
46      for i := 0; i < len(l); i++ {
47          if l[i] != nil {
48              fmt.Printf(
49                  "|%3v|%10v|%10v|%10v|\n",
50                  i, l[i].doneNumber.Sub(start), l[i].doneCvv.Sub(start),
                        l[i].doneExp.Sub(start),
51              )
52          }
53      }
54      fmt.Printf("%v%36v%v\n", "+", strings.Repeat("-", 36), "+")
55
56      fmt.Printf("%38v\n", aurora.BgRed("IDLE␣TIME"))
57      fmt.Printf("%v%36v%v\n", "+", strings.Repeat("-", 36), "+")
58      fmt.Printf("|%3v|%32v|\n", aurora.Green("N"), aurora.Green("Idle␣time"))
59      fmt.Printf("%v%36v%v\n", "+", strings.Repeat("-", 36), "+")
60      for i := 0; i < len(l)-1; i++ {
61          fdtime += l[i+1].startNumber.Sub(start) - l[i].doneNumber.Sub(start)
62          sdtime += l[i+1].startCvv.Sub(start) - l[i].doneCvv.Sub(start)
63          tdtime += l[i+1].startExp.Sub(start) - l[i].doneExp.Sub(start)
64      }
65      ldtime := []time.Duration{fdtime, sdtime, tdtime}
66      for i := 0; i < 3; i++ {
67          fmt.Printf("|%3v|%32v|\n", i+1, ldtime[i])
68      }
69      fmt.Printf("%v%36v%v\n", "+", strings.Repeat("-", 36), "+")
70
71      if logCreditCard {
72          fmt.Printf("\n%v:\n", aurora.Magenta("Generated␣data"))
73          for i := range l {
74              fmt.Printf("CreditCard:␣%d\nNumber:␣%s\nCVV:␣%s\nExpiration␣date:␣%s\n\n",
```

```
                l[i].num+1, l[i].number, l[i].cvv, l[i].exp)
        }
    }
}
```

Вывод

Была разработана реализация конвейерных вычислений.

4 Исследовательская часть

В данном разделе приведены примеры работы и анализ характеристик разработанного программного обеспечения.

4.1 Пример работы

Демонстрация работы программы приведена на рисунке 4.1.

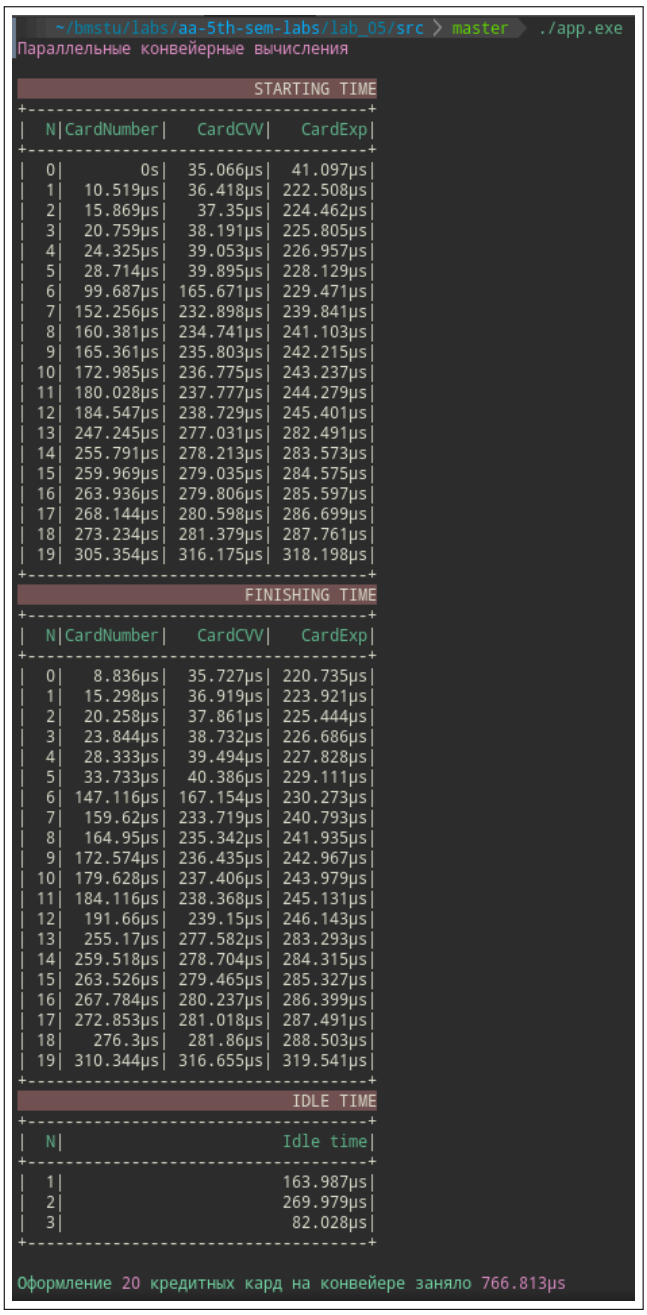

Рисунок 4.1 – Демонстрация работы алгоритмов сортировки

4.2 Технические характеристики

Технические характеристики устройства, на котором выполнялось тестирование:

- Операционная система: Manjaro [2] Linux [3] 20.1 64-битная.
- Память: 16 ГБ.
- Процессор: AMD Ryzen™ 7 3700U [4] ЦПУ @ 2.30ГГц

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

4.3 Время выполнения алгоритмов

Алгоритмы тестировались при помощи написания «бенчмарков» [5], предоставялемых 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

```
1  package conv
2
3  import "testing"
4
5  func BenchmarkConveyor10(b *testing.B) {
6      N := 10
7      ch := make(chan int)
8
9      for i := 0; i < b.N; i++ {
10         Conveyor(N, ch)
11         <-ch
12     }
```

```
}

func BenchmarkConveyor20(b *testing.B) {
    N := 20

    for i := 0; i < b.N; i++ {
        ch := make(chan int)
        Conveyor(N, ch)
        <-ch
    }
}
```

На рисунке 4.2 показаны результаты работы бенчмарков.

```
~/bmstu/labs/aa-5th-sem-labs/lab_05/src/conv > master  go test -bench .
goos: linux
goarch: amd64
BenchmarkConveyor10-8              15121             89238 ns/op
BenchmarkConveyor20-8              10000            150496 ns/op
BenchmarkConveyor30-8              10000            215068 ns/op
BenchmarkConveyor40-8               8052            267778 ns/op
BenchmarkConveyor50-8               7832            331040 ns/op
PASS
ok      _/home/hackfeed/bmstu/labs/aa-5th-sem-labs/lab_05/src/conv      11.955s
```

Рисунок 4.2 – Демонстрация работы бенчмарков

Вывод

Асинхронные конвейерные вычисления позволяют организовать непрерывную обработку данных, что позволяет выиграть время в задачах, где требуется обработка больших объемов данных за малый промежуток времени.

Заключение

В ходе выполнения работы была достигнута цель выполнены все поставленные задачи:

- рассмотреть и изучить асинхронную конвейерную обработку данных;
- реализовать систему конвейерных вычислений с количеством линий не меньше трех;
- на основании проделанной работы сделать выводы.

Асинхронные конвейерные вычисления позволяют организовать непрерывную обработку данных, что позволяет выиграть время в задачах, где требуется обработка больших объемов данных за малый промежуток времени.

Литература

[1] The Go Programming Language [Электронный ресурс]. Режим доступа: https://golang.org/ (дата обращения: 11.09.2020).

[2] Manjaro – enjoy the simplicity [Электронный ресурс]. Режим доступа: https://manjaro.org/ (дата обращения: 14.09.2020).

[3] Linux – Википедия [Электронный ресурс]. Режим доступа: https://ru.wikipedia.org/wiki/Linux (дата обращения: 14.09.2020).

[4] Мобильный процессор AMD Ryzen™ 7 3700U с графикой Radeon™ RX Vega 10 [Электронный ресурс]. Режим доступа: https://www.amd.com/ru/products/apu/amd-ryzen-7-3700u (дата обращения: 14.09.2020).

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