

МГТУ им. БАУМАНА

ЛАБОРАТОРНАЯ РАБОТА №5

По курсу: "АНАЛИЗ АЛГОРИТМОВ"

## **Конвейерная обработка**

Работу выполнил: Мокеев Даниил, ИУ7-54

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

*Москва, 2019*

# Оглавление

<b>Введение</b>	<b>2</b>
<b>1 Аналитическая часть</b>	<b>3</b>
1.1 Параллельное программирование . . . . .	3
1.1.1 Организация взаимодействия параллельных потоков . . . . .	4
1.2 Вывод . . . . .	4
<b>2 Конструкторская часть</b>	<b>5</b>
2.1 Распараллеливание программы . . . . .	5
2.2 Вывод . . . . .	5
<b>3 Технологическая часть</b>	<b>6</b>
3.1 Выбор ЯП . . . . .	6
3.2 Описание структуры ПО . . . . .	6
3.3 Сведения о модулях программы . . . . .	6
3.4 Листинг кода алгоритмов . . . . .	7
3.5 Вывод . . . . .	9
<b>4 Исследовательская часть</b>	<b>10</b>
4.1 Примеры работы . . . . .	10
4.2 Постановка эксперимента . . . . .	12
4.2.1 Заключение экспериментальной части . . . . .	14
<b>Заключение</b>	<b>15</b>
<b>Список литературы</b>	<b>15</b>

# Введение

Цель работы: изучение возможности конвейерной обработки и использование такого подхода на практике. Необходимо сравнить времени работы алгоритма на нескольких потоках 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. В терминах программирования ленты конвейера представлены функциями, выполняющими над неким набором данных операции и передающие 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 я решил выбрать торты - на первой линии конвейера замешивается тесто, на второй наносят глазурь, на третьей декорируют.

## 1.1 Параллельное программирование

При использовании многопроцессорных вычислительных систем с общей памятью 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 очередность и синхронизация доступа обеспечивается на аппаратном уровне). Многопроцессорные системы подобного типа обычно именуются симметричными мультипроцессорами (symmetric multiprocessors, SMP).

Перечисленному выше набору предположений удовлетворяют также активно развиваемые в последнее время многоядерные процессоры, 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 вычислений достаточно ограничены, и второй подход является преобладающим. При этом для разработки параллельных программ могут применяться как новые алгоритмические языки, ориентированные на параллельное программирование, так и уже имеющиеся языки, расширенные некоторым набором операторов для параллельных вычислений.

Широко используемый подход состоит и в применении тех или иных библиотек, обеспечивающих определенный программный интерфейс (application programming interface, API) для разработки параллельных программ. В рамках такого подхода наиболее известны Windows Thread API. Однако первый способ применим только для ОС семейства Microsoft Windows, а второй вариант API является достаточно трудоемким для использования и имеет низкоуровневый характер [?].

### **1.1.1 Организация взаимодействия параллельных потоков**

Потоки исполняются в общем адресном пространстве параллельной программы. Как результат, взаимодействие параллельных потоков можно организовать через использование общих данных, являющихся доступными для всех потоков. Наиболее простая ситуация состоит в использовании общих данных только для чтения. В случае же, когда общие данные могут изменяться несколькими потоками, необходимы специальные усилия для организации правильного взаимодействия.

## **1.2 Вывод**

Была рассмотрена конвейерная обработка данных, технология параллельного программирования и организация взаимодействия параллельных потоков.

## 2 | Конструкторская часть

### Требования к вводу:

На ввод подается целое число - желаемое количество изготовленных экземпляров

### Требования к программе:

- вывод статистики обработанных экземпляров;
- при матрицах неправильных размеров программа не должна аварийно завершаться.

## 2.1 Распараллеливание программы

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

## 2.2 Вывод

В данном разделе была рассмотрена схема алгоритма и способ ее распараллеливания.

## 3 | Технологическая часть

### 3.1 Выбор ЯП

Я выбрал в качестве языка программирования Golang, потому как он достаточно удобен, быстр и успешно использует концепции мультипоточного программирования.

Время работы алгоритмов было замерено с помощью функции `Now()` из библиотеки `time`.

### 3.2 Описание структуры ПО

В данном разделе будет рассмотрена структура ПО (Рис. 3.1)

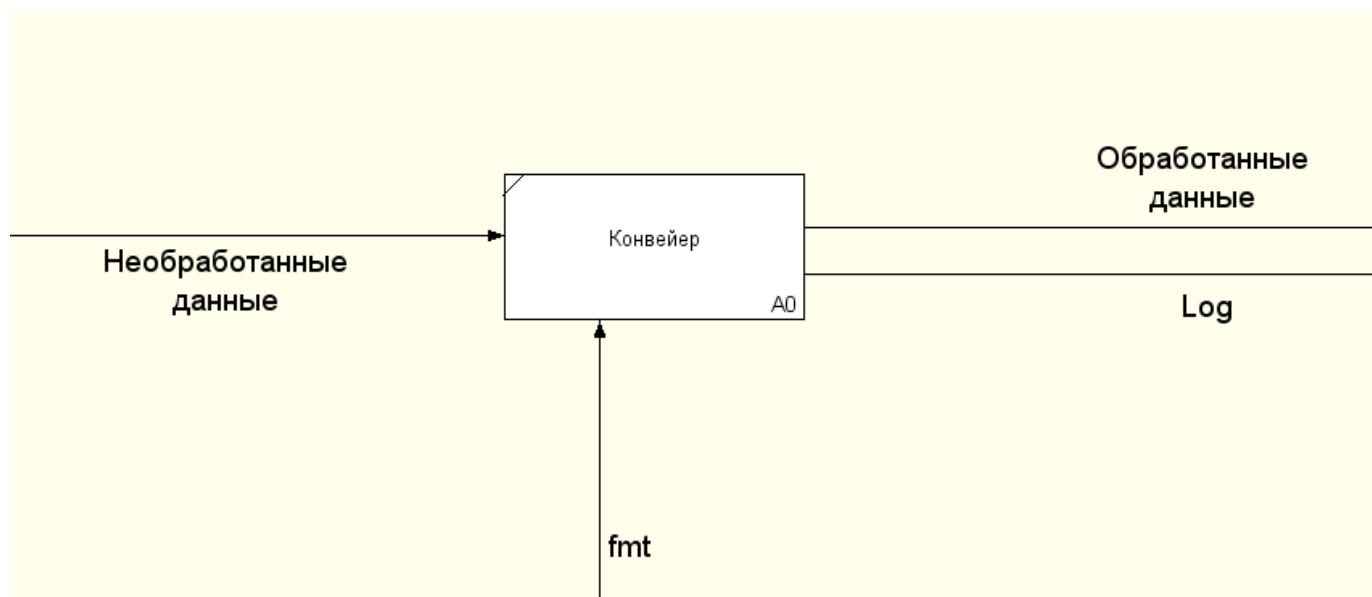


Рис. 3.1: Функциональная схема умножения матриц (IDEF0 диаграмма 1 уровня)

### 3.3 Сведения о модулях программы

Программа состоит из:

- `lab05.go`- главный файл программы, в котором располагается точка входа в программу.

- `linear.go` - файл, содержащий функцию линейной обработки данных.

## 3.4 Листинг кода алгоритмов

В данном разделе будут приведены листинги кода параллельной реализации конвейера (Листинг 3.1) и линейной реализации (Листинг 3.2)

Листинг 3.1: Параллельный конвейер

```

1 func conv(amount int, wait chan int) *queue{
2     uno := make(chan *cake, 5)
3     dos := make(chan *cake, 5)
4     tres := make(chan *cake, 5)
5     line := new_queue(amount)
6     first := func() {
7         for{
8             select{
9                 case a := <- uno:
10                a.dough = true
11
12                a.started_dough = time.Now()
13                took_dough := 200
14                time.Sleep(time.Duration(took_dough) * time.Millisecond)
15
16                a.finished_dough = time.Now()
17                dos <- a
18            }
19        }
20    }
21
22    second := func() {
23        for{
24            select{
25                case a := <- dos:
26                //fmt.Printf("Cake num %d started topping\n", a.num)
27                a.topping = true
28
29                a.started_topping = time.Now()
30                took_topping := 200
31                time.Sleep(time.Duration(took_topping) * time.Millisecond)
32
33                a.finished_topping = time.Now()
34                tres <- a
35            }

```



```

36     }
37 }
38
39 third := func(){
40     for{
41         select{
42             case a := <- tres:
43                 //fmt.Printf("Cake num %d started decor\n", a.num)
44                 a.decor = true
45
46                 a.started_decor = time.Now()
47                 took_decor := 200
48                 time.Sleep(time.Duration(took_decor) * time.Millisecond)
49
50                 a.finished_decor = time.Now()
51                 line.push(a)
52                 if (a.num == amount){
53                     wait <- 0 }
54
55             }
56         }
57     }
58
59     go first()
60     go second()
61     go third()
62     for i:=0; i<=amount; i++){
63         a := new(cake)
64         a.num = i
65         uno <- a
66     }
67     return line
68 }

```

Листинг 3.2: Линейный конвейер

```

1 func linear (amount int)*queue{
2     queue_for_topping := new_queue(amount)
3     queue_for_decor := new_queue(amount)
4     finished := new_queue(amount)
5     i:= 0
6     for ; i!=-1;{
7         a := new(cake)
8         a.num = i

```

```

9      first(a, queue_for_topping)
10     if queue_for_topping.last >= 0{
11         second(queue_for_topping.pop(), queue_for_decor)
12     }
13     if queue_for_decor.last >= 0{
14         third(queue_for_decor.pop(), finished)
15     }
16     if finished.waiting[len(finished.waiting)-1] != nil{
17         return finished}
18     i+=1
19 }
20 return finished
21 }

```

## 3.5 Вывод

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

## 4 | Исследовательская часть

Был проведен замер времени работы алгоритмов с использованием разного количества изготавливаемых изделий. Исследования были проведены на процессоре Intel Core i5-6200U с двумя физическими и 4 логическими ядрами. На каждом конвейере время производства занимало 0.2сек

### 4.1 Примеры работы

В данном разделе будут приведены примеры работы программы - лог конвейерной обработки (Рис. 4.1) и лог линейной обработки (Рис 4.2)

```

0 0s 200.4626ms 400.9273ms
1 200.4626ms 400.9273ms 601.3926ms
2 400.9273ms 601.3926ms 801.854ms
3 601.3926ms 801.854ms 1.001885s
4 801.854ms 1.001885s 1.2020108s
5 1.001885s 1.2020108s 1.4024753s
6 1.2020108s 1.4024753s 1.602908s
7 1.4024753s 1.6039013s 1.8050533s
8 1.6039013s 1.8050533s 2.0060843s
9 1.8050533s 2.0060843s 2.2066001s
10 2.0060843s 2.2066001s 2.4074376s
11 2.2066001s 2.4074376s 2.6082281s
12 2.4074376s 2.6082281s 2.8086935s
13 2.6082281s 2.8086935s 3.00944s
14 2.8086935s 3.00944s 3.2181985s
15 3.00944s 3.2181985s 3.4212007s
16 3.2181985s 3.4212007s 3.6219956s
17 3.4212007s 3.6219956s 3.8221841s
18 3.6219956s 3.8221841s 4.0225792s
19 3.8221841s 4.0225792s 4.2229721s
Finishing time
0 200.4626ms 400.9273ms 601.3926ms
1 400.9273ms 601.3926ms 801.854ms
2 601.3926ms 801.854ms 1.001885s
3 801.854ms 1.001885s 1.2020108s
4 1.001885s 1.2020108s 1.4024753s
5 1.2020108s 1.4024753s 1.602908s
6 1.4024753s 1.602908s 1.8030593s
7 1.6039013s 1.8050533s 2.0060843s
8 1.8050533s 2.0060843s 2.2066001s
9 2.0060843s 2.2066001s 2.4074376s
10 2.2066001s 2.4074376s 2.6082281s
11 2.4074376s 2.6082281s 2.8086935s
12 2.6082281s 2.8086935s 3.00944s
13 2.8086935s 3.00944s 3.2181985s
14 3.00944s 3.2181985s 3.4212007s
15 3.2181985s 3.4206581s 3.6219956s
16 3.4212007s 3.6219956s 3.8221841s
17 3.6219956s 3.8221841s 4.0225792s
18 3.8221841s 4.0225792s 4.2229721s
19 4.0225792s 4.2229721s 4.4233456s
Линии простаивали
0s 1.5359ms 1.994ms
To make 20 cakes conv took 4.6551032s

```

Рис. 4.1: Пример работы программы - конвейер на потоках

```

Starting time
0 0s 200.5331ms 400.9968ms
1 601.9809ms 802.705ms 1.0031436s
2 1.2036448s 1.4044298s 1.6051013s
3 1.8058874s 2.0059551s 2.2064136s
4 2.4069601s 2.6074672s 2.8078041s
5 3.0081254s 3.2087826s 3.4092384s
6 3.6101888s 3.8102751s 4.0108308s
7 4.2116898s 4.4122859s 4.6127492s
8 4.8129637s 5.0134974s 5.2139179s
9 5.4145939s 5.6153454s 5.815996s
10 6.0169075s 6.2173643s 6.4181604s
11 6.6190029s 6.8199675s 7.0204288s
12 7.2210341s 7.4214978s 7.6218754s
13 7.8220547s 8.0227762s 8.2229746s
14 8.4239258s 8.6247147s 8.8253891s
15 9.025996s 9.2267982s 9.4272576s
16 9.6279732s 9.8281415s 10.0285573s
17 10.2289995s 10.429593s 10.6300686s
18 10.8305201s 11.0310962s 11.2314502s
19 11.4319878s 11.6328814s 11.8335109s
Finishing time
0 200.5331ms 400.9968ms 601.9809ms
1 802.705ms 1.0031436s 1.2036448s
2 1.4044298s 1.6051013s 1.8058874s
3 2.0059551s 2.2064136s 2.4069601s
4 2.6074672s 2.8078041s 3.0081254s
5 3.2087826s 3.4092384s 3.6101888s
6 3.8102751s 4.0108308s 4.2116898s
7 4.4122859s 4.6127492s 4.8129637s
8 5.0134974s 5.2139179s 5.4145939s
9 5.6153454s 5.815996s 6.0169075s
10 6.2173643s 6.4181604s 6.6190029s
11 6.8199675s 7.0204288s 7.2210341s
12 7.4214978s 7.6218754s 7.8220547s
13 8.0227762s 8.2229746s 8.4239258s
14 8.6247147s 8.8253891s 9.025996s
15 9.2267982s 9.4272576s 9.6279732s
16 9.8281415s 10.0285573s 10.2289995s
17 10.429593s 10.6300686s 10.8305201s
18 11.0310962s 11.2314502s 11.4319878s
19 11.6328814s 11.8335109s 12.0343013s
Линии простаивали
7.6212104s 7.6232206s 7.6204314s
To make 20 cakes linear conv took 12.0343013s

```

Рис. 4.2: Пример работы программы - линейная обработка данных и результат

## 4.2 Постановка эксперимента

Проведем сравнение для каждого из алгоритмов. Для замера времени будем использовать функцию `time.Now()`

Сравним результаты для линейной обработки данных и распараллеленной:

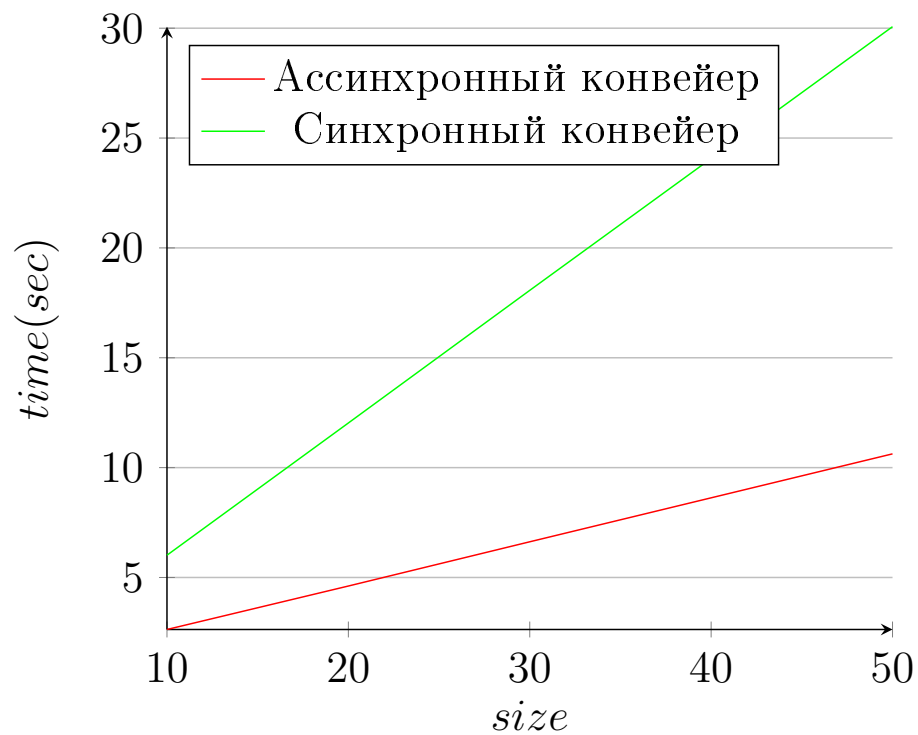


Рис. 4.3: Сравнение параллельного и обычного алгоритмов

#### 4.2.1 Заключение экспериментальной части

Эксперимент 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.

# Заключение

В ходе лабораторной работы были изучены возможности параллельных вычислений, реализован алгоритм конвейерной обработки данных с помощью параллельных вычислений.

Было проведено сравнение синхронной версии того же алгоритма и асинхронной. Выяснилось, что при использовании потоков, время работы алгоритма не просто сокращается, но и снижается скорость роста времени при увеличении числа изготавливаемых экземпляров.



# Литература

- [1] Меднов В.П., Бондаренко Е.П. Транспортные, распределительные и рабочие конвейеры. М., 1970.
- [2] Конвейерное производство[Электронный ресурс] - режим доступа <https://dic.academic.ru/dic.nsf/ruwiki/1526795>
- [3] Конвейерный метод производства Генри Форда[Электронный ресурс] - режим доступа <https://ropecon.ru/305-konveiernyi-metod-proizvodstva-genri-forda.html>
- [4] И. В. Белоусов(2006), Матрицы и определители, учебное пособие по линейной алгебре, с. 1 - 16