



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.

# Оглавление

<b>Введение</b>	<b>3</b>
<b>1 Аналитическая часть</b>	<b>4</b>
1.1 Параллельное программирование . . . . .	4
1.2 Организация взаимодействия параллельных потоков . . . . .	5
<b>2 Конструкторская часть</b>	<b>6</b>
2.1 Распараллеливание программы . . . . .	6
<b>3 Технологическая часть</b>	<b>7</b>
3.1 Требования к ПО . . . . .	7
3.2 Структура ПО . . . . .	7
3.3 Средства реализации . . . . .	7
3.4 Листинг кода . . . . .	8
<b>4 Исследовательская часть</b>	<b>11</b>
4.1 Пример работы . . . . .	11
4.2 Технические характеристики . . . . .	12
4.3 Время выполнения алгоритмов . . . . .	13
4.4 Производительность алгоритмов . . . . .	14
<b>Заключение</b>	<b>16</b>
<b>Литература</b>	<b>17</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 [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 я решил выбрать торты - на первой линии конвейера происходит подготовка, на второй выпекание, на третьей проводят финальные работы по приготовлению.

## 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 (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.

## 1.2 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.

## 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.

### 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 были 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.

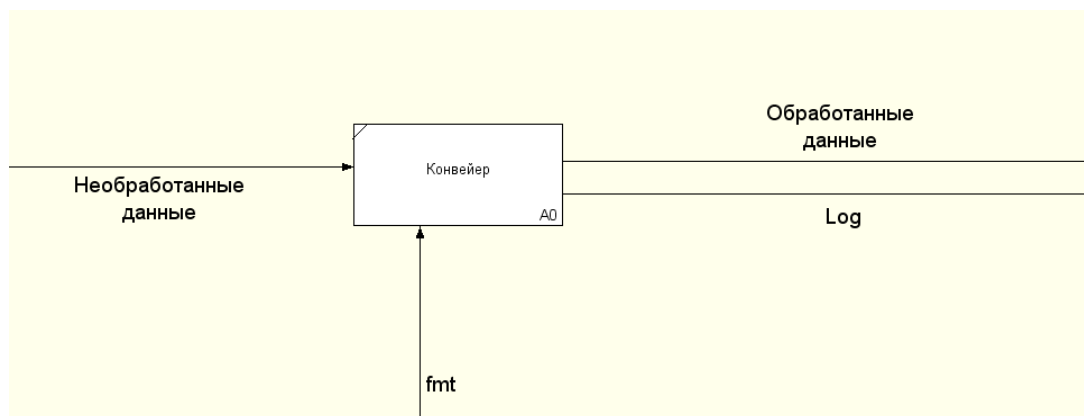


Рисунок 3.1 – Структура ПО

### 3.3 Средства реализации

В качестве языка программирования для реализации данной лабораторной работы был выбран многопоточный язык GO [2]. Данный выбор обусловлен моим желанием расширить свои знания в области применения данного языка. Так же данный язык предоставляет средства тестирования разработанного ПО.

Время работы алгоритмов было замерено с помощью функции `Now()` из библиотеки `Time` [3].

## 3.4 Листинг кода

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

Листинг 3.1 – Реализация параллельного конвейера

```
1 func first(cake *Cake, qBake *Queue) {
2     cake.prepare = true
3
4     cake.sPrepare = time.Now()
5     time.Sleep(time.Duration(200) * time.Millisecond)
6     cake.fPrepare = time.Now()
7
8     qBake.push(cake)
9 }
10
11 func second(cake *Cake, qFinalize *Queue) {
12     cake.bake = true
13
14     cake.sBake = time.Now()
15     time.Sleep(time.Duration(200) * time.Millisecond)
16     cake.fBake = time.Now()
17
18     qFinalize.push(cake)
19 }
20
21 func third(cake *Cake, finished *Queue) {
22     cake.finalize = true
23
24     cake.sFinalize = time.Now()
25     time.Sleep(time.Duration(200) * time.Millisecond)
26     cake.fFinalize = time.Now()
27
28     finished.push(cake)
29 }
30
31 func Parallel(amount int, wait chan int) *Queue {
32     f := make(chan *Cake, 5)
33     s := make(chan *Cake, 5)
34     t := make(chan *Cake, 5)
35     line := createQueue(amount)
36     first := func() {
37         for {
38             select {
39                 case cake := <-f:
40                     cake.prepare = true
```



```

41
42         cake.sPrepare = time.Now()
43         time.Sleep(time.Duration(200) * time.Millisecond)
44         cake.fPrepare = time.Now()
45
46         s <- cake
47     }
48 }
49 }
50
51 second := func() {
52     for {
53         select {
54             case cake := <-s:
55                 cake.bake = true
56
57                 cake.sBake = time.Now()
58                 time.Sleep(time.Duration(200) * time.Millisecond)
59                 cake.fBake = time.Now()
60
61                 t <- cake
62             }
63         }
64     }
65
66 third := func() {
67     for {
68         select {
69             case cake := <-t:
70                 cake.finalize = true
71
72                 cake.sFinalize = time.Now()
73                 time.Sleep(time.Duration(200) * time.Millisecond)
74                 cake.fFinalize = time.Now()
75
76                 line.push(cake)
77                 if cake.num == amount {
78                     wait <- 0
79                 }
80             }
81         }
82     }
83 }
84
85 go first()
86 go second()
87 go third()
88 for i := 0; i <= amount; i++ {

```

```

89     cake := new(Cake)
90     cake.num = i
91     f <- cake
92 }
93
94 return line
95 }

```

### Листинг 3.2 – Реализация линейного конвейера

```

1 func Linear(amount int) *Queue {
2     qBake := createQueue(amount)
3     qFinalize := createQueue(amount)
4     finished := createQueue(amount)
5     i := 0
6     for i != -1 {
7         cake := new(Cake)
8         cake.num = i
9         first(cake, qBake)
10        if qBake.l >= 0 {
11            second(qBake.pop(), qFinalize)
12        }
13        if qFinalize.l >= 0 {
14            third(qFinalize.pop(), finished)
15        }
16        if finished.q[len(finished.q)-1] != nil {
17            return finished
18        }
19        i++
20    }
21    return finished
22 }

```

## 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

```
Starting time
0 0s 200.144714ms 400.28121ms
1 600.388958ms 800.604383ms 1.000741799s
2 1.20091758s 1.401060009s 1.601157857s
3 1.801318756s 2.001536867s 2.201813516s
4 2.401955708s 2.602137149s 2.802157628s
5 3.002357437s 3.202620621s 3.402856554s
6 3.6030625s 3.803083892s 4.003244519s
7 4.20350003s 4.403676205s 4.603893421s
8 4.804164785s 5.004393986s 5.204529845s
9 5.404715617s 5.604938126s 5.805037081s
10 6.005205095s 6.205415106s 6.405566079s
11 6.605775836s 6.805870048s 7.005969839s
12 7.206117519s 7.406260801s 7.60627864s
13 7.806435435s 8.006565911s 8.206824536s
14 8.407132804s 8.607376165s 8.807396399s
15 9.007553204s 9.207820268s 9.408083184s
16 9.60814326s 9.808285144s 10.008545651s
17 10.208720585s 10.408851809s 10.60910758s
18 10.809327482s 11.009595153s 11.20977201s
19 11.409908228s 11.610136458s 11.810239365s
Finishing time
0 200.144389ms 400.280817ms 600.384656ms
1 800.604046ms 1.000741351s 1.200914747s
2 1.40105967s 1.601156235s 1.801316277s
3 2.001536535s 2.201813026s 2.401952769s
4 2.602136806s 2.802157209s 3.002353833s
5 3.202620318s 3.402856107s 3.603059792s
6 3.803083539s 4.003244138s 4.203497203s
7 4.403675878s 4.60389298s 4.804160771s
8 5.004393663s 5.204529435s 5.404713146s
9 5.604937769s 5.805036616s 6.005201974s
10 6.205414805s 6.40556562s 6.605772756s
11 6.805869657s 7.005969315s 7.206114021s
12 7.406260504s 7.606278138s 7.806430907s
13 8.006565433s 8.206824144s 8.407129944s
14 8.607375777s 8.807395959s 9.007548729s
15 9.207819878s 9.40808271s 9.608140106s
16 9.808284687s 10.008545158s 10.208717878s
17 10.408851496s 10.609107176s 10.809324255s
18 11.009594801s 11.209771608s 11.409904472s
19 11.61013532s 11.810238884s 12.010424044s
Линии простаивали
7.606471168s 7.606980359s 7.606577267s
20 тортов на линейном конвейере: 12.010427795s
```

Рисунок 4.1 – Лог работы линейной обработки

```

Starting time
0 0s 200.126252ms 400.268455ms
1 200.097644ms 400.321417ms 600.379957ms
2 400.298386ms 600.378128ms 800.601504ms
3 600.316017ms 800.59514ms 1.000933113s
4 800.583795ms 1.000862272s 1.201202429s
5 1.000857479s 1.201201359s 1.401329506s
6 1.201132095s 1.401328807s 1.601599248s
7 1.401322806s 1.601596953s 1.801738063s
8 1.601585399s 1.801761497s 2.00186227s
9 1.801755443s 2.001869896s 2.201997671s
10 2.001865442s 2.202004147s 2.402260245s
11 2.202000504s 2.402267119s 2.602389395s
12 2.402263379s 2.602396442s 2.802657616s
13 2.602392307s 2.802664454s 3.00272356s
14 2.802660519s 3.002728362s 3.202990204s
15 3.00272671s 3.202993472s 3.40312585s
16 3.202992766s 3.403129034s 3.603394306s
17 3.403128306s 3.60339763s 3.803484914s
18 3.603396875s 3.803488501s 4.003743934s
19 3.803487716s 4.003748074s 4.204033988s
Finishing time
0 200.094576ms 400.263031ms 600.321875ms
1 400.297321ms 600.376132ms 800.600543ms
2 600.313839ms 800.594637ms 1.000932287s
3 800.581286ms 1.000852643s 1.201106834s
4 1.000856028s 1.201160807s 1.401318403s
5 1.201130088s 1.401327777s 1.601579094s
6 1.401321269s 1.601594575s 1.801733706s
7 1.601583723s 1.801736342s 2.001828436s
8 1.801753269s 2.0018593s 2.201996856s
9 2.001863889s 2.201993867s 2.402259819s
10 2.201999174s 2.402256835s 2.602388942s
11 2.402262057s 2.602385765s 2.802657117s
12 2.60239097s 2.802654055s 3.002723134s
13 2.802659175s 3.002720258s 3.202989738s
14 3.00272512s 3.202986584s 3.403125334s
15 3.202991773s 3.403122182s 3.603393901s
16 3.403127355s 3.60339059s 3.803484425s
17 3.603395895s 3.803481655s 4.003743375s
18 3.80348673s 4.003739722s 4.20403354s
19 4.003746033s 4.204030652s 4.404075551s
Линии простаивали
30.051µs 235.947µs 230.414µs
20 тортов на параллельном конвейере: 4.6047596s

```

Рисунок 4.2 – Лог работы конвейерной обработки

## 4.2 Технические характеристики

Технические характеристики устройства, на котором выполнялось тестирование:

- Операционная система: Kali [4] Linux [5] 5.9.0-kali1-amd64.
- Память: 8 GB.

- Процессор: Intel® Core™ i5-8250U [6] 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» [7], 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

```
1 func BenchmarkParallel10(b *testing.B) {  
2     for i := 0; i < b.N; i++ {  
3         wait := make(chan int)  
4         Parallel(10, wait)  
5         <-wait  
6     }  
7 }
```

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

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.

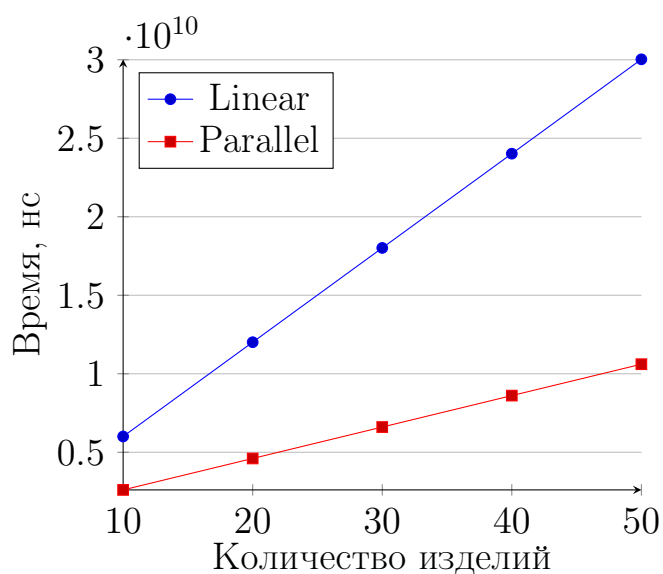


Рисунок 4.3 – Сравнение конвейеров.

```
(root@NebuchadnezzaR) - [~/bmstu-aa/lab5]
# make bench
cd src/conv/ && go test -bench . -benchmem
goos: linux
goarch: amd64
BenchmarkParallel10-8      1      2602784445 ns/op      4800 B/op      43 allocs/op
BenchmarkParallel20-8      1      4604745168 ns/op      6160 B/op      45 allocs/op
BenchmarkParallel30-8      1      6606492569 ns/op      7552 B/op      52 allocs/op
BenchmarkParallel40-8      1      8609241453 ns/op      9328 B/op      63 allocs/op
BenchmarkParallel50-8      1     10609299378 ns/op     11656 B/op      82 allocs/op
BenchmarkLinear10-8        1      6005714661 ns/op      1952 B/op      15 allocs/op
BenchmarkLinear20-8        1     12010312654 ns/op      3792 B/op      25 allocs/op
BenchmarkLinear30-8        1     18014944704 ns/op      5632 B/op      35 allocs/op
BenchmarkLinear40-8        1     24018505373 ns/op      7472 B/op      45 allocs/op
BenchmarkLinear50-8        1     30028223784 ns/op      9360 B/op      55 allocs/op
PASS
ok      _/root/bmstu-aa/lab5/src/conv 123.119s
```

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

# 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] В. П. Меднов Е. П. Бондарев. Транспортные, распределительные и рабочие конвейеры. – М, 1970.
- [2] The Go Programming Language [Электронный ресурс]. Режим доступа: <https://golang.org/> (дата обращения: 09.09.2020).
- [3] testing – Package time [Электронный ресурс]. Режим доступа: <https://golang.org/pkg/time/> (дата обращения: 18.11.2020).
- [4] Our Most Advanced Penetration Testing Distribution, Ever. [Электронный ресурс]. Режим доступа: <https://kali.org/> (дата обращения: 12.09.2020).
- [5] Linux – Википедия [Электронный ресурс]. Режим доступа: <https://ru.wikipedia.org/wiki/Linux> (дата обращения: 12.09.2020).
- [6] 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).
- [7] testing – The Go Programming Language [Электронный ресурс]. Режим доступа: <https://golang.org/pkg/testing/> (дата обращения: 12.09.2020).