



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

Введение	3
1 Аналитическая часть	4
1.1 Параллельное программирование	4
2 Конструкторская часть	6
2.1 Распараллеливание программы	6
3 Технологическая часть	7
3.1 Требования к ПО	7
3.2 Структура ПО	7
3.3 Средства реализации	7
3.4 Листинг кода	8
4 Исследовательская часть	12
4.1 Пример работы	12
4.2 Технические характеристики	13
4.3 Время выполнения алгоритмов	14
4.4 Производительность алгоритмов	15
Заключение	17
Литература	18

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 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 очередность 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 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.

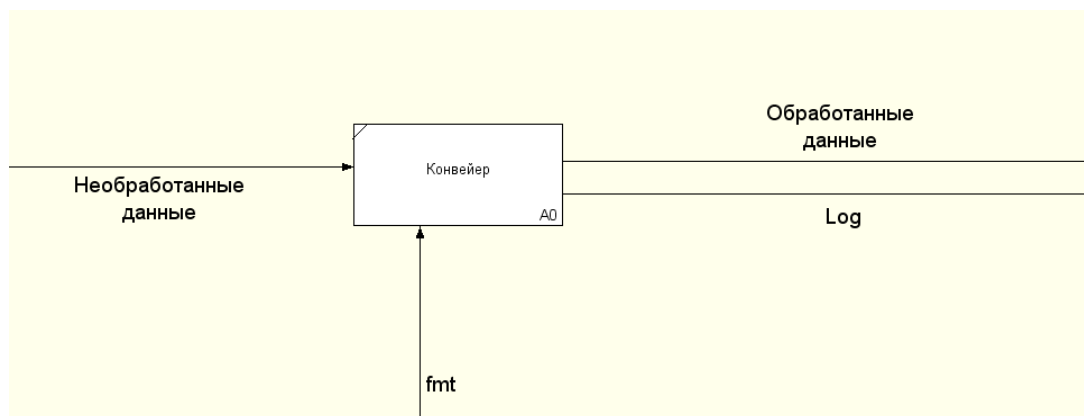


Рисунок 3.1 – Структура ПО

3.3 Средства реализации

В качестве языка программирования для реализации данной лабораторной работы был выбран многопоточный язык GO [7]. 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` [11].

3.4 Листинг кода

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

Листинг 3.1 – Реализация параллельного конвейера

```
1 func Parallel(amount int, wait chan int) *Queue {
2     f := make(chan *Person, 5)
3     s := make(chan *Person, 5)
4     t := make(chan *Person, 5)
5     line := createQueue(amount)
6     name := func() {
7         for {
8             select {
9                 case p := <-f:
10                    p.haveName = true
11
12                    p.startName = time.Now()
13                    randomOps()
14                    p.name = gofakeit.Name()
15                    randomOps()
16                    p.doneName = time.Now()
17
18                    s <- p
19            }
20        }
21    }
22
23    email := func() {
24        for {
25            select {
26                case p := <-s:
27                    p.haveEmail = true
28
29                    p.startEmail = time.Now()
30                    randomOps()
31                    p.email = gofakeit.Email()
32                    randomOps()
33                    p.doneEmail = time.Now()
34
35                    t <- p
36            }
37        }
38    }
39
40    beer := func() {
```



```

41     for {
42         select {
43             case p := <-t:
44                 p.haveBeer = true
45
46                 p.startBeer = time.Now()
47                 randomOps()
48                 p.beer = gofakeit.BeerName()
49                 randomOps()
50                 p.doneBeer = time.Now()
51
52                 line.push(p)
53                 if p.num == amount {
54                     wait <- 0
55                 }
56
57             }
58         }
59     }
60
61     go name()
62     go email()
63     go beer()
64     for i := 0; i <= amount; i++ {
65         p := new(Person)
66         p.num = i
67         f <- p
68     }
69
70     return line
71 }

```

Листинг 3.2 – Реализация линейного конвейера

```

1 func name(p *Person, qEmail *Queue) {
2     p.haveName = true
3
4     p.startName = time.Now()
5     randomOps()
6     p.name = gofakeit.Name()
7     randomOps()
8     p.doneName = time.Now()
9
10    qEmail.push(p)
11 }
12
13 func email(p *Person, qBeer *Queue) {
14     p.haveEmail = true
15

```

```

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

```
Starting time
0 0s 207.964608ms 409.29426ms
1 605.358479ms 799.586491ms 1.02654734s
2 1.284457508s 1.483217584s 1.683007561s
3 1.872493986s 2.063315315s 2.261030611s
4 2.446944816s 2.643997815s 2.83473606s
5 3.030472264s 3.225389469s 3.415820915s
6 3.61489569s 3.807673327s 4.004107202s
7 4.200604067s 4.392928917s 4.592164794s
8 4.786809615s 4.985545807s 5.184715344s
9 5.380600039s 5.581124446s 5.775804414s
10 5.971615142s 6.171602777s 6.366028098s
11 6.564127747s 6.764939661s 6.964802497s
12 7.163244632s 7.360880835s 7.558642178s
13 7.758256966s 7.959896135s 8.158033617s
14 8.353277068s 8.549067442s 8.747790342s
15 8.943298278s 9.143306335s 9.3397803s
16 9.538207093s 9.738545535s 9.936410727s
17 10.134619248s 10.332562891s 10.528128268s
18 10.724876855s 10.922903107s 11.120666573s
19 11.318038905s 11.518191742s 11.755805004s
Finishing time
0 207.964063ms 409.293869ms 605.356201ms
1 799.586137ms 1.02654677s 1.284455542s
2 1.483217213s 1.683006943s 1.872491553s
3 2.063314978s 2.261030231s 2.446942316s
4 2.643997459s 2.834735678s 3.030462958s
5 3.22538913s 3.415820498s 3.614893292s
6 3.807672935s 4.00410685s 4.200601737s
7 4.392928678s 4.59216435s 4.78680765s
8 4.985545555s 5.184714926s 5.380597595s
9 5.581124117s 5.775804017s 5.971612527s
10 6.171602407s 6.366027697s 6.564125051s
11 6.764939306s 6.964802035s 7.163241889s
12 7.360880464s 7.558641744s 7.758254945s
13 7.95989575s 8.158032804s 8.353275457s
14 8.549067152s 8.747790056s 8.943296119s
15 9.143306096s 9.339779917s 9.538204591s
16 9.738544977s 9.936410376s 10.134617222s
17 10.33256258s 10.528127852s 10.724874412s
18 10.922902724s 11.120666124s 11.318036056s
19 11.518191434s 11.755803426s 12.047141665s
Линии простаивали
7.557756677s 7.537172894s 7.561874732s
20 персональнх карточек на линейном конвейере : 12.047144391s
```

Рисунок 4.1 – Лог работы линейной обработки

```

Starting time
0 0s 202.981713ms 308.156465ms
1 202.935594ms 404.120904ms 501.061823ms
2 404.075037ms 600.159524ms 695.070061ms
3 600.111409ms 799.339396ms 898.768361ms
4 799.294685ms 995.604286ms 1.103093322s
5 995.555095ms 1.208137888s 1.32281667s
6 1.208090258s 1.43715003s 1.539756225s
7 1.437144806s 1.639143726s 1.731607406s
8 1.639072693s 1.83744403s 1.937932918s
9 1.837386492s 2.033623988s 2.131128778s
10 2.03357413s 2.252528524s 2.358164113s
11 2.25248318s 2.471360278s 2.57754999s
12 2.471312419s 2.666539309s 2.773085672s
13 2.666533456s 2.881942847s 2.997352291s
14 2.881879049s 3.098000794s 3.209849533s
15 3.09799566s 3.304780382s 3.411033078s
16 3.304674971s 3.540250418s 3.645482467s
17 3.540166306s 3.741884057s 3.83974726s
18 3.741803035s 3.943627128s 4.054367825s
19 3.943556549s 4.160432946s 4.265581637s
Finishing time
0 202.929814ms 308.150522ms 403.106371ms
1 404.069969ms 501.056454ms 602.050433ms
2 600.105692ms 695.062445ms 799.985155ms
3 799.289313ms 898.762511ms 996.882372ms
4 995.546051ms 1.10308753s 1.208366217s
5 1.208085609s 1.322811442s 1.43660399s
6 1.437139394s 1.53975213s 1.638690075s
7 1.639068395s 1.731600956s 1.834245003s
8 1.8373834s 1.937927426s 2.033444547s
9 2.033569539s 2.13112284s 2.252961937s
10 2.252476329s 2.358159272s 2.473047574s
11 2.471307652s 2.577541003s 2.6682813s
12 2.666528395s 2.773080066s 2.88055737s
13 2.881873635s 2.997344996s 3.097657438s
14 3.097990198s 3.209843524s 3.304847368s
15 3.304671607s 3.411027029s 3.538739039s
16 3.540161441s 3.645475717s 3.741245457s
17 3.741798709s 3.839741624s 3.943629934s
18 3.943547193s 4.05436126s 4.158142759s
19 4.160366641s 4.265574881s 4.380001345s
Линии простаивали
102.489µs 1.980161708s 1.980965091s
20 персональных карточек на параллельном конвейере: 4.584718225s

```

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

```
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.

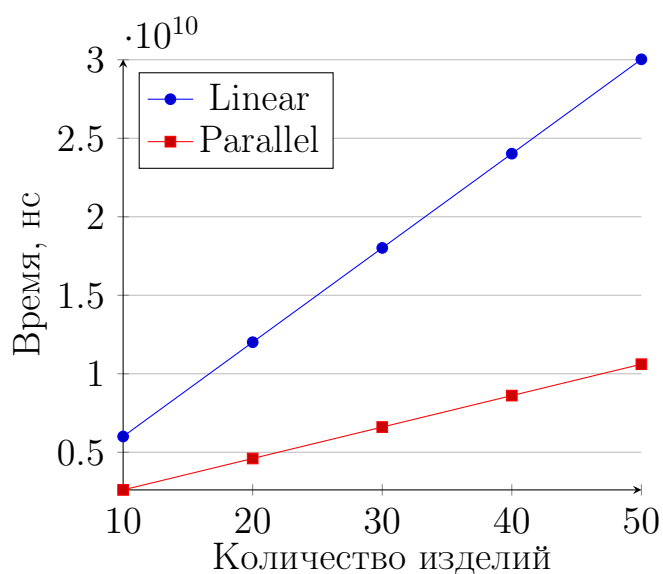


Рисунок 4.3 – Сравнение конвейеров.

```
(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, 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).