



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”

Студент _____ Миронов Григорий Александрович _____
Группа _____ ИУ7-53Б _____
Название предприятия _____ МГТУ им. Н. Э. Баумана, каф. ИУ7 _____
Тема _____ Конвейерная обработка _____

Студент	_____	Миронов Г. А.
	(подпись, дата)	(фамилия, и.о.)
Преподаватель	_____	Волкова Л. Л.
	(подпись, дата)	(фамилия, и.о.)

Москва
2021 г.

Оглавление

Введение	3
1 Аналитическая часть	4
1.1 Конвейерная обработка	4
1.2 Предметная область	4
1.3 Вывод	4
2 Конструкторская часть	5
2.1 Схемы	5
2.2 Вывод	6
3 Технологическая часть	7
3.1 Выбор средств реализации	7
3.2 Требования к программному обеспечению	7
3.3 Сведения о модулях программы	7
3.4 Вывод	11
4 Экспериментальная часть	12
4.1 Технические характеристики	12
4.2 Тестирование	12
4.3 Временные характеристики	13
4.4 Вывод	15
Заключение	16
Литература	17

Введение

В данной лабораторной работе будут рассмотрены конвейерные на примере вычисления хеш-сумм файлов в файловой системе.

Конвейерный принцип обработки [1] - способ организации вычислений, используемый в современных процессорах и контроллерах с целью повышения их производительности, путем увеличения числа инструкций, выполняемых в единицу времени (эксплуатация параллелизма на уровне инструкций).

Основная идея конвейерной обработки заключается в разделении подлежащей выполнению функции на более мелкие части и выделении для каждой из них отдельного блока аппаратуры. Прирост производительности достигается именно за счет параллельного выполнения частей более сложной функции.

Целью данной работы является реализация и изучение конвейерной обработки.

Для достижения данной цели необходимо решить следующие задачи:

- изучить основные методы организации конвейерных вычислений;
- реализовать конвейер;
- провести экспериментальные замеры временных характеристик реализованного конвейера.

1 Аналитическая часть

1.1 Конвейерная обработка

В рамках данной лабораторной работы конвейерной обработкой будет считать так называемую "синхронную" конвейерную обработку [2]. В конвейерах данного типа отсутствуют очереди между различными "лентами" конвейера. В связи с этим, на конвейерах, организованных подобным образом, часть лент может оказаться заблокированной, т.к. для продолжения выполнения кода необходимо дождаться освобождения последующей ленты конвейера.

Тем не менее, теоритически подобный подход является менее затратным по памяти, т.к. не использует дополнительных структур для реализации очередей.

1.2 Предметная область

В качестве алгоритма, реализованного для распределения на конвейере, было выбрано вычисление кеш-сумм каждого из файлов в заданном каталоге файловой системы. Данный алгоритм состоит из 3-х этапов:

- получение полного пути файла;
- вычисление кеш-суммы (алгоритм md5 [3]) содержимого файла;
- агрегация полученных результатов и сохранение в результирующую структуру.

1.3 Вывод

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

2 Конструкторская часть

В данном разделе будут рассмотрены схемы алгоритмов умножения матриц и модель вычислений.

2.1 Схемы

На рисунке 2.1 представлена схема организации конвейерных вычислений на примере конвейера с тремя лентами.

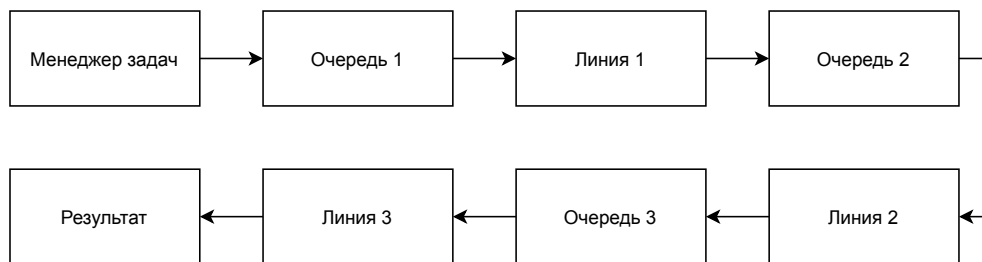


Рисунок 2.1 – Схема организации конвейерных вычислений

Заметим, так же, что вычисление хеш-суммы файла не зависит от ранее вычисленных значений. В связи с этим, можно реализовать многопоточное вычисление хеш-сумм файлов, посредством так называемых fan-out и fan-in подходов.

Таким образом, схема 2.1 может быть преобразована следующим образом:

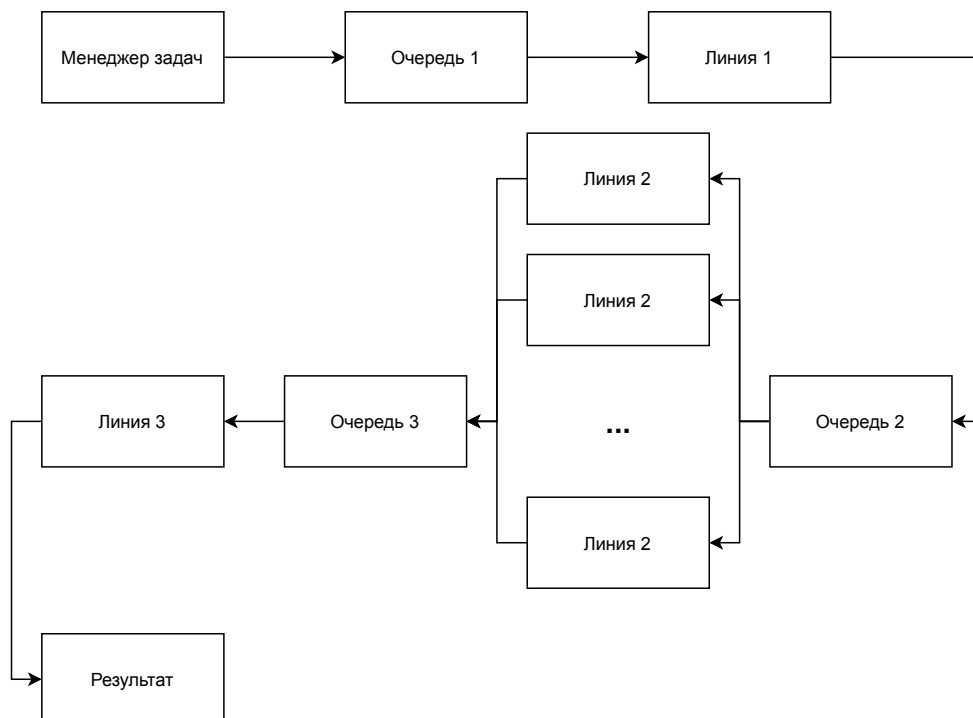


Рисунок 2.2 – Схема организации конвейерных вычислений

Отдельно стоит отметить, что в алгоритме, рассматриваемом в рамках данной лабораторной работы, отсутствует первая очередь конвейера, т.к. первая лента и является основным поставщиком задач для обработки.

2.2 Вывод

На основе теоретических данных, полученных из аналитического раздела, была построена схема организации конвейерных вычислений на примере конвейере с тремя лентами (рис. 2.1).

3 Технологическая часть

3.1 Выбор средств реализации

В качестве языка программирования для реализации данной лабораторной работы был выбран язык Golang [4]. Данный выбор обусловлен тем, что я имею некоторый опыт разработки на нем, а так же наличием у языка встроенных высокоточных средств тестирования и анализа разработанного ПО.

3.2 Требования к программному обеспечению

Входные данные: путь до файла или директории в файловой системе.

Выходные данные: упорядоченный по имени файла набор строк, содержащих имя файла и соответствующий кеш md5.

3.3 Сведения о модулях программы

Данная программа разбита на модули:

- `main.c` - файл, содержащий точку входа в программу;
- `types.go` - файл, содержащий определение пользовательских типов данных;
- `conveyer.go` - файл, содержащий определение структуры конвейера;
- `filewalker.go` - файл, содержащий реализацию первой ленты конвейера;
- `digester.go` - файл, содержащий реализацию второй ленты конвейера;
- `md5.go` - файл, содержащий реализацию третьей ленты конвейера;

На листингах 3.1 – 3.6 представлен код программы.

Листинг 3.1 – Основной файл программы `main`

```
1 func main() {  
2     conveyor := md5conveyor.NewConveyor(30)  
3     start := time.Now()  
4     data, err := conveyor.Md5All(os.Args[1])  
5     if err != nil {  
6         fmt.Println(err)  
7         return  
8     }  
9     process(data, time.Since(start).Nanoseconds())  
10 }
```

Листинг 3.2 – Определение пользовательских типов данных

```
1 type fwOutput struct {
2     path string
3
4     processTime time.Duration
5     queueStart time.Time
6 }
7
8 type dOutput struct {
9     path string
10    sum [md5.Size]byte
11    err error
12
13    filewalker time.Duration
14    queue time.Duration
15    processTime time.Duration
16    md5Start time.Time
17 }
18
19 type Output struct {
20     Path string
21     Sum [md5.Size]byte
22
23     Filewalker time.Duration
24     DigesterQueue time.Duration
25     Digester time.Duration
26     Queue time.Duration
27 }
```

Листинг 3.3 – Определение структуры конвейера

```
1 type Conveyor struct {
2     numDigesters int
3 }
4
5 func NewConveyor(numDigesters int) Conveyor {
6     return Conveyor{numDigesters}
7 }
```


Листинг 3.4 – Лента обхода файловой системы

```

1 func walkFiles(done <-chan struct{}, root string) (<-chan fwOutput, <-chan error) {
2     paths := make(chan fwOutput)
3     errc := make(chan error, 1)
4     go func() {
5         defer close(paths)
6         start := time.Now()
7         errc <- filepath.Walk(root, func(path string, info os.FileInfo, err error) error {
8             if err != nil {
9                 return err
10            }
11            if !info.Mode().IsRegular() {
12                return nil
13            }
14            select {
15            case paths <- fwOutput{path, time.Since(start), time.Now()}:
16                start = time.Now()
17            case <-done:
18                return errors.New("walk canceled")
19            }
20            return nil
21        })
22    }()
23    return paths, errc
24 }

```

Листинг 3.5 – Лента вычисления хеш-суммы конкретного файла

```

1 func digester(done <-chan struct{}, paths <-chan fwOutput, c chan<- dOutput) {
2     for path := range paths {
3         start := time.Now()
4         data, err := ioutil.ReadFile(path.path)
5         select {
6         case c <- dOutput{
7             path.path,
8             md5.Sum(data),
9             err,
10            path.processTime,
11            start.Sub(path.queueStart),
12            time.Since(start),
13            time.Now(),
14        }:
15            start = time.Now()
16        case <-done:
17            return
18        }
19    }
20 }

```

Листинг 3.6 – Запуск конвейера

```

1 func (c *Conveyor) Md5All(root string) ([]Output, error) {
2     done := make(chan struct{})
3     defer close(done)
4
5     paths, errc := walkFiles(done, root)
6
7     out := make(chan dOutput)
8     var wg sync.WaitGroup
9     wg.Add(c.numDigesters)
10    for i := 0; i < c.numDigesters; i++ {
11        go func() {
12            digester(done, paths, out)
13            wg.Done()
14        }()
15    }
16    go func() {
17        wg.Wait()
18        close(out)
19    }()
20
21    m := make([]Output, 0)
22    for r := range out {
23        if r.err != nil {
24            return nil, r.err
25        }
26        m = append(m, Output{
27            r.path,
28            r.sum,
29            r.filewalker,
30            r.queue,
31            r.processTime,
32            time.Since(r.md5Start),
33        })
34    }
35
36    if err := <-errc; err != nil {
37        return nil, err
38    }
39    return m, nil
40 }

```

3.4 Вывод

В данном разделе был реализован вышеописанный алгоритм.

Было разработано программное обеспечение, удовлетворяющее предъявляемым требованиям. Так же были представлены соответствующие листинги 3.1 – 3.6 с кодом программы.

4 Экспериментальная часть

В данном разделе будет проведено функциональное тестирование разработанного программного обеспечения. Так же будет произведено измерение временных характеристик каждого из реализованных алгоритмов.

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялось исследование:

- Процессор: Intel Core™ i5-8250U [5] CPU @ 1.60GHz.
- Память: 32 GiB.
- Операционная система: Manjaro [6] Linux [7] 21.1.4 64-bit.

Исследование 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.

4.2 Тестирование

В данном разделе будет приведена таблица с тестами (таблица ??).

Таблица 4.1 – Таблица тестов

Путь	Файлы	Хеш-сумма
./main.go	./main.go	8bbb6d9acffca1a5f9f51e2152b1f535
./src	go.mod lab_05 main.go md5conveyor/conveyer.go md5conveyor/digester.go md5conveyor/filewalker.go md5conveyor/md5.go md5conveyor/types.go process.go	feca2534ea9898769375d13a1e00c287 2eb11a3416254f86320142b58e1896ff 8bbb6d9acffca1a5f9f51e2152b1f535 2224737183186219e0a8564a5b1724cb 3b85b3efbe4291c2e79efee842ce6439 d9a373972260d3883eb7159ad8a77532 5f3bf37395cb7e3a236ba096b0e316d6 2715f98eda4714f34aee1646f5514f63 58300a8b36b9e1d01d54d842ed219418

При проведении функционального тестирования, полученные результаты работы программы совпали с ожидаемыми. Таким образом, функциональное тестирование пройдено успешно.

4.3 Временные характеристики

Так как в реализуемом конвейере реализуются fan-out и fan-in подходы, особый интерес представляет зависимость временных характеристик системы от количества параллельных сопрограмм.

Для сравнения возьмем конвейеры с соответствующим числом сопрограмм из набора [1, 2, 4, 8, ..., 64].

На момент тестирования в файловой системе машины, на которой проводились тесты, насчитывалось 23447250 файлов различного типа и размера. В ходе проведения эксперимента данная величина оставалась неизменной.

Так как в общем случае вычисление хеш-суммы файла является достаточно короткой задачей, воспользуемся усреднением массового эксперимента. Для этого вычислим среднее арифметическое значение временных ресурсов, затраченных на выполнение алгоритма, для n запусков. Сравнение произведем при $n = 100$.

В первую очередь, интерес представляет временные характеристики реализованного конвейера, в сравнении с линейной реализацией алгоритма. В качестве временных характеристик линейной реализации примем общее время, которое задачи находились в процессе обработки конвейером (т.е. суммарное время простоя и обработки на каждой из лент конвейера).

Результаты замеров по результатам экспериментов приведены в Таблице 4.2.

Таблица 4.2 – Сравнение общего и реального времени выполнения алгоритма

Количество сопрограмм, шт.	Общее время работы системы, нс	Реальное время работы системы, нс
1	3e+12	1.6e+12
2	2.7e+12	9.8e+11
4	3.3e+12	7.4e+11
8	4.5e+12	5.9e+11
16	7.1e+12	5.4e+11
32	1.2e+13	5.2e+11
64	2.3e+13	5.1e+11

Из данных, приведенных в таблице 4.2, можно сделать вывод, что конвейерная организация вычислений является более эффективной, чем линейная (вплоть до двух порядков, при количестве сопрограмм равном 64).

Отдельно стоит отметить факт того, что, при запуске всего одной сопрограммы, общее время работы в двое превосходит реальное время работы конвейера. Такой результат является ожидаемым, так как, с ростом числа задач, обрабатываемых конвейером, пропорционально увеличивается общее время ожидания и простоя.

На Рисунке 4.1 отображены временные характеристики работы конвейера из Таблицы 4.2.

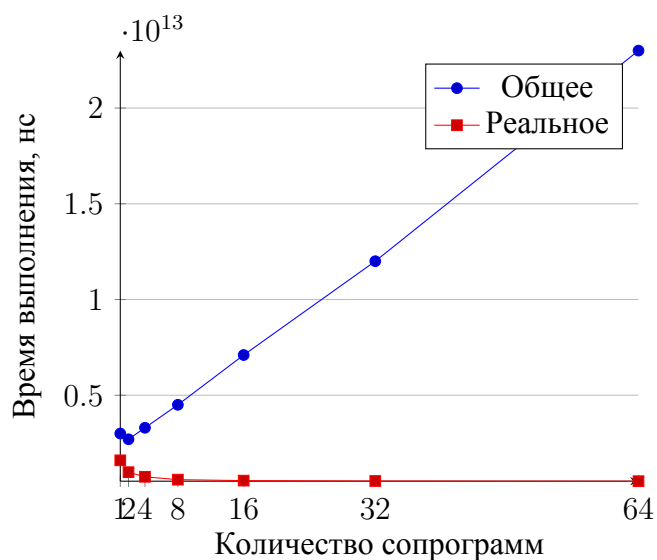


Рисунок 4.1 – Сравнение общего и реального времени выполнения алгоритма

Из Рисунка 4.2 следует, что, несмотря на увеличивающееся общее время выполнения алгоритма, реальное время выполнения алгоритма уменьшается (вплоть до 3 раз при увеличении от 1 до 64 сопрограмм).

Так как сопрограммы в языке программирования `golang` реализуются в рамках одного потока, максимальная производительность конвейера достигается при значениях больших количества логических ядер конкретной машины.

Отдельно сравним среднее время выполнения каждой из стадий конвейера. Данные, полученные в ходе эксперимента, приведены в Таблице 4.3. Т.к. третья лента конвейера является завершающей, остальные ленты не оказывают влияния на время обработки ею задачи. В связи с этим, среднее время обработки задачи третьей лентой будет исключено из сравнения.

Таблица 4.3 – Среднее время выполнения различных стадий конвейера

Количество сопрограмм, шт.	Среднее время			
	Лента 1, нс	Очередь 2, нс	Лента 2, нс	Очередь 3, нс
1	4.7e+03	5.3e+04	6.3e+04	8e+03
2	5.6e+03	3.2e+04	6.9e+04	7.7e+03
4	7.3e+03	2.9e+04	9.7e+04	8.6e+03
8	7.8e+03	3.2e+04	1.4e+05	1.2e+04
16	8.3e+03	7.8e+04	1.9e+05	3e+04
32	9.5e+03	1.7e+05	2.8e+05	6.9e+04
64	1e+04	3.4e+05	4.8e+05	1.5e+05

Из таблицы 4.3 следует, что, с ростом числа сопрограмм, растёт среднее время выполнения каждой из стадий конвейера. Это связано с дополнительными затратами из-за многопоточной реализации второй ленты конвейера.

На Рисунке 4.2 отображены усредненные временные характеристики выполнения каждой из стадий конвейера.

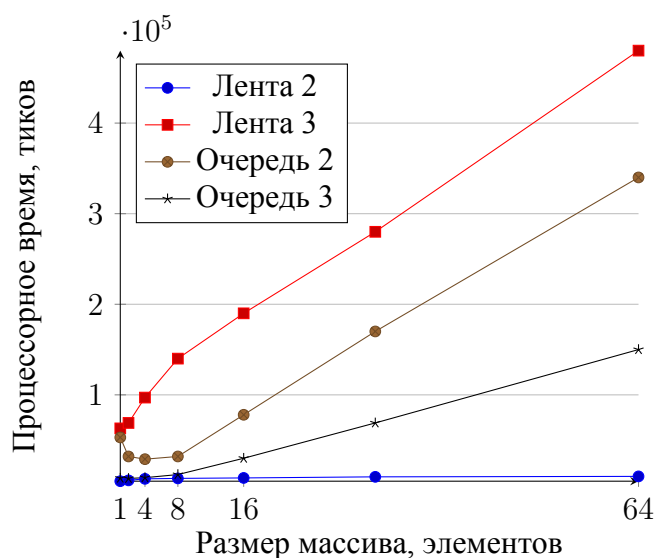


Рисунок 4.2 – Усредненные временные характеристики выполнения каждой из стадий конвейера

Из Рисунка 4.2 следует, что увеличивается не только время ожидания в очереди, но и время обработки задачи конкретной лентой конвейера. Связано это с синхронной реализацией конвейера - очереди между лентами являются небуфферизированными, т.е. пропускают лишь одну задачу за раз, блокируя остальные ленты.

В связи с ростом числа сопрограмм, 3 становится так называемым "бутылочным горлышком" конвейера. Для решения этой проблемы используются очереди динамической размерности (в процессе выполнения программы они могут расширяться, чтобы хранить большее число задач, не приводя к блокировке остальных стадий конвейера).

4.4 Вывод

В данном разделе было произведено экспериментальное исследование количества затраченного времени вышеизложенным алгоритмом.

Наиболее эффективной оказалась реализация конвейера с большим (64 шт.) числом сопрограмм в рамках третьей ленты. Несмотря на увеличение времени простоя системы (в 10 раз, в сравнении с реализацией с 1 сопрограммой), реальное время работы конвейера снизилось в 3 раза (в сравнении с реализацией с 1 сопрограммой).

Для оптимизации существующего решения стоит использовать очереди динамической размерности, чтобы предотвратить блокировку частей конвейера друг другом. Т.к. такая реализация потребует дополнительных затрат памяти, следует использовать конвейер с фиксированным размером очереди между лентами в тех случаях, когда существуют жесткие ограничения по памяти.

Заключение

В данной лабораторной работе была конвейерная организация вычислений.

Среди рассмотренных алгоритмов наиболее эффективным по времени является параллельный алгоритм умножения матриц по строкам, так как в нем отсутствуют лишние обращения к памяти.

В связи с вышесказанным, параллельный алгоритм умножения по строкам является предпочтительным при обработке больших матриц в многопоточном окружении, однако, при работе с матрицами малых размерностей (меньше 64), стандартный алгоритм умножения становится более эффективным в связи с дополнительными затратами на организацию параллельности вычислений (создание потоков, организация совместного доступа к ресурсам).

В рамках выполнения работы решены следующие задачи.

- изучены основные методы организации конвейерных вычислений;
- реализован конвейер;
- проведены экспериментальные замеры временных характеристик реализованного конвейера.

Литература

- [1] Р.А. Волков А.Н. Гнутов В.К. Дьячков и др. Конвейеры. Справочник. Машиностроение, Ленинградское отделение, 1984.
- [2] К. Кормен Т. Лейзерсон Ч. Ривест Р. Штайн. Алгоритмы: построение и анализ. – М.: Вильямс, 2013. с. 1296.
- [3] rfc1321 - IETF Tools [Электронный ресурс]. Режим доступа: <https://datatracker.ietf.org/doc/html/rfc1321> (дата обращения: 24.10.2021).
- [4] The Go Programming Language [Электронный ресурс]. Режим доступа: <https://golang.org/> (дата обращения: 24.10.2021).
- [5] Процессор Intel® Core™ i5-8250U [Электронный ресурс]. Режим доступа: <https://ark.intel.com/content/www/ru/ru/ark/products/124967/intel-core-i5-8250u-processor-6m-cache-up-to-3-40-ghz.html> (дата обращения: 24.10.2021).
- [6] Manjaro - enjoy the simplicity [Электронный ресурс]. Режим доступа: <https://manjaro.org/> (дата обращения: 24.10.2021).
- [7] LINUX.ORG.RU – Русская информация об ОС Linux [Электронный ресурс]. Режим доступа: <https://www.linux.org.ru/> (дата обращения: 24.10.2021).