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

по курсу «Анализ Алгоритмов»

на тему: «Конвейерная обработка»

Студент ИУ7-53Б _____ Миронов Г. А.
(Группа) (Подпись, дата) (И. О. Фамилия)

Преподаватель _____ Волкова Л. Л.
(Подпись, дата) (И. О. Фамилия)

2021 г.

СОДЕРЖАНИЕ

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 ковейерной 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:

- исследовать основы конвейерных вычислений;
- исследовать основные методы организации конвейерных вычислений;
- сравнить существующие методы организации конвейерных вычислений;
- привести схемы рассматриваемых алгоритмов, а именно:
  - схему конвейера, содержащего 3 ленты;
  - схему конвейера, содержащего 3 ленты и реализующего `fan-in`–`fan-out` подходы;
- описать использующиеся структуры данных;
- описать структуру разрабатываемого програмного обеспечения;
- определить средства программной реализации;
- определить требования к программному обеспечению;
- привести сведения о модулях программы;
- провести тестирование реализованного программного обеспечения;

- провести экспериментальные замеры временных характеристик реализованного конвейера.

1 Аналитическая часть

В данном разделе рассматривается понятие конвейерной обработки.

Так же, рассматривается предметная область лабораторной работы:

- описание решаемой задачи;
- описание каждой из выделенных стадий реализуемого алгоритма решения данной задачи.

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.

Входными данными для программного обеспечения служит путь до файла или директории в файловой системе.

Выходными данными являются:

- упорядоченный по имени список обработанных файлой и соответствующие им хеш–суммы;
- усредненные временные характеристики конвейера: среднее время обработки заявки каждой из лент, а так же - усредненное время ожидания в каждой из очередей конвейера.

На программное обеспечения накладываются следующие ограничения:

- входные данные должны быть корреткными т.е.:
  - введенный путь должен быть корреткным;
  - введенному пути должен существовать файл/директория
- программа должна иметь соответствующие права времени выполнения для чтения файлов.

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.

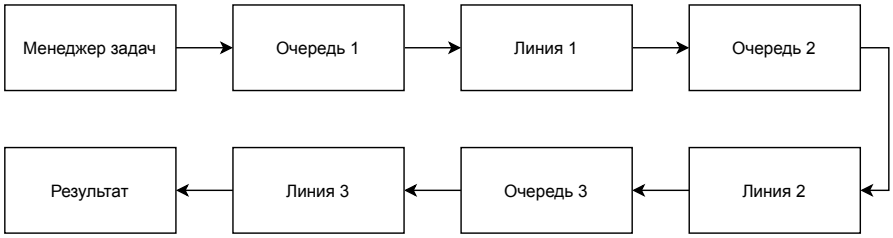

Рисунок 2.1 – Схема организации конвейерных вычислений

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:

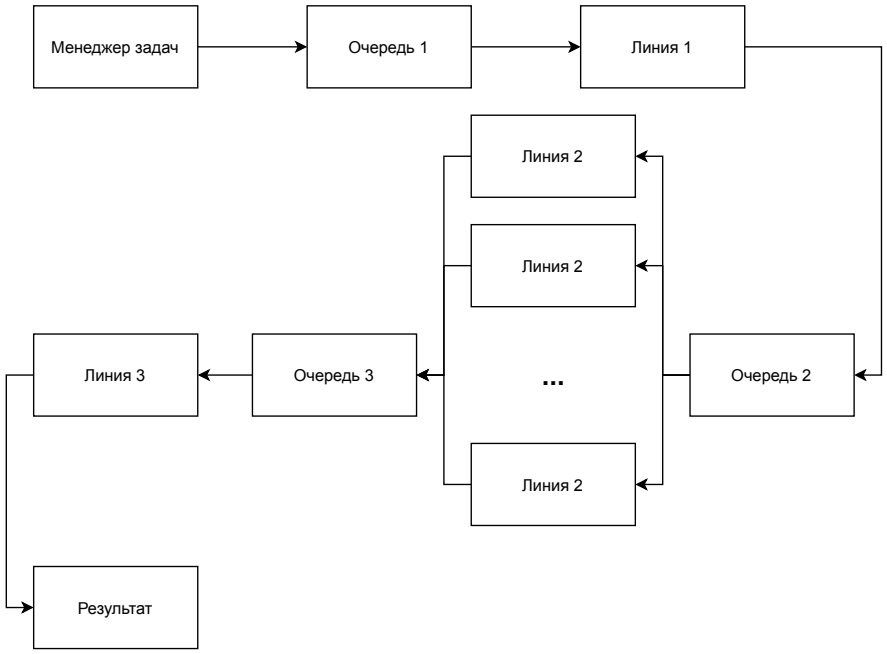

Рисунок 2.2 – Схема организации конвейерных вычислений

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 Описание структуры программного обеспечения

На Рисунке 2.3 представлена `idef0`–диаграмма работы описанного выше конвейера.

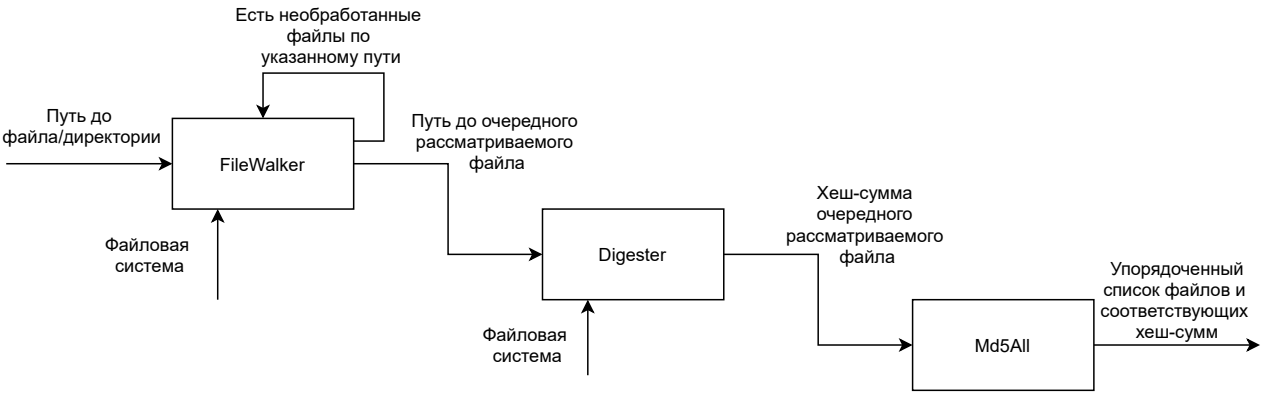

Рисунок 2.3 – Idef0–диаграмма работы конвейера

2.3 Описание структур данных

Для реализации конвейерных вычислений, введем некоторые пользовательские типы данных:

- `fwOutput` – структура, описывающая результат обработки задачи первой лентой конвейера;
- `dOutput` – структура, описывающая результат обработки задачи второй лентой конвейера;
- `Output` – структура, описывающая результат обработки задачи конвейером.

Рассмотрим каждый из введенных пользовательских данных.

2.3.1 Описание пользовательского типа данных fwOutput

Листинг 2.1 – Определение пользовательских типов данных. Часть 1

```
type fwOutput struct {
    path string

    processTime time.Duration
    queueStart  time.Time
}
```

Здесь:

- `path` – строка, содержащая путь до рассматриваемого в задаче файла;
- `processTime` – длительность обработки задачи первой лентой конвейера;
- `queueStart` – момент попадания задачи в очередь второй ленты.

2.3.2 Описание пользовательского типа данных dOutput

Листинг 2.2 – Определение пользовательских типов данных. Часть 2

```
type dOutput struct {
    path string
    sum  [md5.Size]byte
    err  error

    filewalker  time.Duration
    queue       time.Duration
    processTime time.Duration
    md5Start    time.Time
}
```

Здесь:

- `path` – строка, содержащая путь до рассматриваемого в задаче файла;
- `sum` – `md5`–хеш содержимого файла, обрабатываемого в рамках задачи;
- `err` – поле, содержащее, при наличии таковой, описание ошибки обработки задачи;

- `filewalker` – длительность обработки задачи первой лентой конвейере;
- `queue` – длительность ожидания в очереди второй ленты конвейера;
- `processTime` – длительность обработки задачи второй лентой конвейера;
- `queueStart` – момент попадания задачи в очередь третьей ленты.

2.3.3 Описание пользовательского типа данных Output

Листинг 2.3 – Определение пользовательских типов данных. Часть 3

```
type Output struct {
    Path string
    Sum  [md5.Size]byte

    Filewalker    time.Duration
    DigesterQueue time.Duration
    Digester      time.Duration
    Queue         time.Duration
}
```

Здесь:

- `Path` – строка, содержащая путь до рассматриваемого в задаче файла;
- `Sum` – `md5`–хеш содержимого файла, обрабатываемого в рамках задачи;
- `Filewalker` – длительность обработки задачи первой лентой конвейере;
- `DigesterQueue` – длительность ожидания в очереди второй ленты конвейера;
- `Digester` – длительность обработки задачи второй лентой конвейера;
- `Queue` – lkbntkmyjcnmдлительность ожидания в очереди 3ей ленты.

2.4 Вывод

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

Так же, было приведено описание пользовательских типов данных, вводимых в рамках реализации конвейерных вычислений (Листинги 2.1 – 2.3).

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, содерящих 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 даных;
- `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

```
func main() {
    conveyor := md5conveyor.NewConveyor(30)
    start := time.Now()
    data, err := conveyor.Md5All(os.Args[1])
    if err != nil {
        fmt.Println(err)
        return
    }
    process(data, time.Since(start).Nanoseconds())
}
```

Листинг 3.2 – Определение пользовательских типов данных

```
type fwOutput struct {
    path string

    processTime time.Duration
    queueStart  time.Time
}

type dOutput struct {
    path string
    sum  [md5.Size]byte
    err  error

    filewalker  time.Duration
    queue       time.Duration
    processTime time.Duration
    md5Start    time.Time
}

type Output struct {
    Path string
    Sum  [md5.Size]byte

    Filewalker    time.Duration
    DigesterQueue time.Duration
    Digester      time.Duration
    Queue         time.Duration
}
```

Листинг 3.3 – Определение структуры конвейера

```
type Conveyor struct {
    numDigesters int
}

func NewConveyor(numDigesters int) Conveyor {
    return Conveyor{numDigesters}
}
```

Листинг 3.4 – Лента обхода файловой системы

```
func walkFiles(done <-chan struct{}, root string) (<-chan
   fwOutput, <-chan error) {
    paths := make(chan fwOutput)
    errc := make(chan error, 1)
    go func() {
        defer close(paths)
        start := time.Now()
        errc <- filepath.Walk(root, func(path string, info os.
           FileInfo, err error) error {
            if err != nil {
                return err
            }
            if !info.Mode().IsRegular() {
                return nil
            }
            select {
            case paths <- fwOutput{path, time.Since(start), time.
               Now()}:
                start = time.Now()
            case <-done:
                return errors.New("walk␣canceled")
            }
            return nil
        })
    }()
    return paths, errc
}
```

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

```
func (c *Conveyor) Md5All(root string) ([]Output, error) {
    done := make(chan struct{})
    defer close(done)

    paths, errc := walkFiles(done, root)

    out := make(chan dOutput)
    var wg sync.WaitGroup
    wg.Add(c.numDigesters)
    for i := 0; i < c.numDigesters; i++ {
        go func() {
            digester(done, paths, out)
            wg.Done()
        }()
    }
    go func() {
        wg.Wait()
        close(out)
    }()

    m := make([]Output, 0)
    for r := range out {
        if r.err != nil {
            return nil, r.err
        }
        m = append(m, Output{
            r.path,
            r.sum,
            r.filewalker,
            r.queue,
            r.processTime,
            time.Since(r.md5Start),
        })
    }

    if err := <-errc; err != nil {
        return nil, err
    }
    return m, nil
}
```

3.4 Тестирование

В рамках данной лабораторной работы будет проведено функциональное тестирование реализованного программного обеспечения.

Выделим основные классы эквивалентности для тестирования:

- входными данными является путь до файла;
- входными данными является путь до директории.

В Таблице 3.1 приведены тесты для указанных классов эквивалентности.

Таблица 3.1 – Таблица тестов

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

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

3.5 Вывод

В данном разделе был реализован вышеописанный алголритм.

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

Было прроведено функциональное тестирование разработанного программного обеспечения. Так же были приведены классы эквивалентности для тестирования.

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 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 4.1.

Таблица 4.1 – 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 4.1, модно 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 зарактеристики работы конвейера из Таблицы 4.1.

Из Рисунка 4.1 следует, что, несмотря на увеличивающеесф общее время выполнения алгоритма, реальное время выполнения алгоритма уменьшается (вплоть до 3 раз при увелечении от 1 до 64 сопрограмм).

Так как сопрограммы в языке программирование `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 4.2. Т.к. третья лента конвейера является завершающей, остальные ленты не оказы-

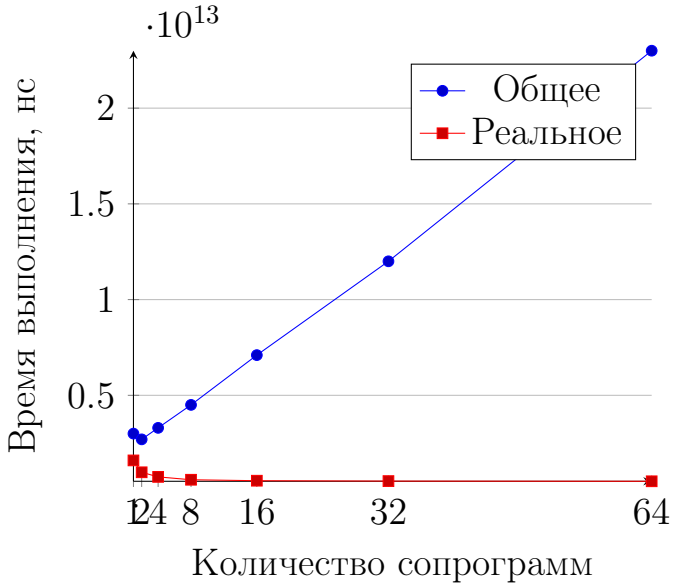

Рисунок 4.1 – Сравнение общего и реального времени выполнения алгоритма

вают влияния на время обработки ею задачи. В связи с этим, среднее время обработки задачи третьей лентой будет исключено из сравнения.

Таблица 4.2 – 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 4.2 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 харкатеристики 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 небуфферoизированными, т.е. пропускают лишь одну задачу за раз, блокируя остальные ленты.

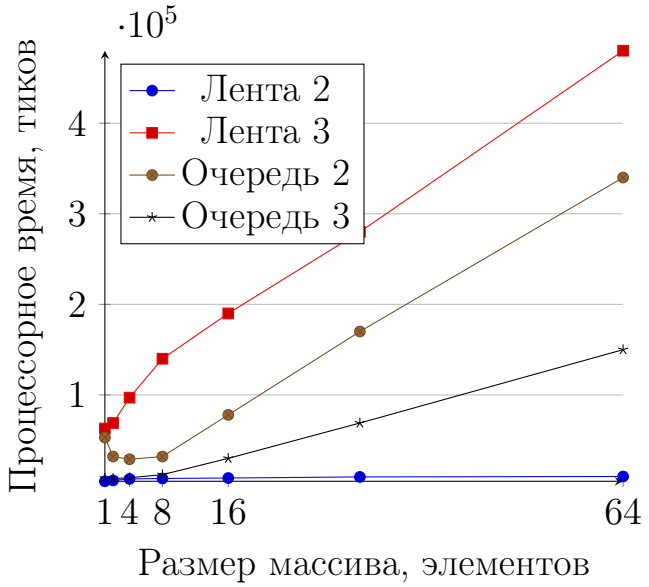

Рисунок 4.2 – Усредненные временные харкатеристики выполнения каждой из стадий конвейера

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

В Таблице 4.3 представлена временная трассировка выполнения для первых шести задач в конвейере. Прочерк означает, что данная задача не обрабатываласб указанной лентой. Такой результат является корректным, например, для директории - хеш-сумма может быть вычислена только у файла, в связи с чем, обрабатывать директорию на второй и третьей ленте бессмысленно.

Из данных, приведенных в Таблице 4.3 следует, что обработка задач действительно происходит асинхронно - во время работы одной ленты, другие выполняют свои функции. Кроме того, видно, что в данной конфигурации конвейера при малом количестве задач в конвейере (в данном случае - шесть) не возникают простои каких-дибо лент.

4.3 Вывод

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

Наиболее эффективной оказалась реализация конвейера с большим (64

Таблица 4.3 – Среднее время выполнения различных стадий конвейера

Лента	**Задача**	**Начало, с**	**Конец, с**
1	1	0.3284e-03	0.3321e-03
2	1	-	-
3	1	-	-
1	2	0.3978e-03	0.4123e-03
2	2	0.5911e-03	0.6701e-03
3	2	0.7028e-03	0.1072e-02
1	3	0.4432e-03	0.5894e-03
2	3	0.6393e-03	0.5559e-02
3	3	0.5603e-02	0.5736e-02
1	4	0.6275e-03	0.6451e-03
2	4	0.6528e-03	0.1092e-02
1	5	0.6983e-03	0.7818e-03
2	5	-	-
3	5	-	-
1	6	0.7952e-03	0.8452e-03
2	6	0.9025e-03	0.9130e-03
3	6	0.1100e-02	0.1106e-02

шт.) числом сопрограмм в рамках третей 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 существующейго 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 наиболе эффективным по времени яляется параллельный алгоритм умножения матриц по строкам, так как в нем отсутствуют лишние обращения к памяти.

В связи с вышуказанным, 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.

- исследованы основы конвейерных вычислений;
- исследованы основные методы организации конвейерных вычислений;
- проведено сравнение существующих методов организации конвейерных вычислений;
- приведены схемы рассматриваемых алгоритмов, а именно:
  - схема конвейера, содержащего 3 ленты;
  - схема конвейера, содержащего 3 ленты и реализующего `fan-in`–`fan-out` подходы;
- описаны использующиеся структуры данных;
- описана структура разрабатываемого програмного обеспечения;
- определены средства программной реализации;
- определены требования к программному обеспечению;
- приведены сведения о модулях программы;
- проведено тестирование реализованного программного обеспечения;

- проведены экспериментальные замеры временных характеристик реализованного конвейера.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. *Р.А. Волков А.Н. Гнутов В. Д. и. д.* Конвейеры. Справочник. — Машиностроение, Ленинградское отделение, 1984.
2. *К. К. Т. Л. Ч. Р. Р. Ш.* 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).