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 «Анализы Алгоритмов»

на тему: «Организация асинхронного взаимодействия потоков вычисления на примере конвейерных вычислений»

Студент группы ИУ7-56Б _____ (Подпись, дата) Мамврийский И. С. (Фамилия И.О.)

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

Преподаватель _____ (Подпись, дата) Строганов Д. В.. (Фамилия И.О.)

Москва — 2023 г.

Содержание

Введение

Сегодня компьютерам требуется производить все более трудоемкие вычисления на больших объемах данных. При этом предъявляются требования к скорости вычислений.

При обработке больших объемов данных часто возникает ситуация, когда каждых набор данных необходимо обработать последовательно несколькими алгоритмами. Так как каждый набор может быть обработан независимого от другого имеется возможность распараллелить вычисления. В таком случае реализуют конвейерную обработку данных, когда каждый алгоритм выполняется на отдельной ленте, запущенной в отдельном потоке, а результат каждого этапа используется в качестве входных данных для следующего.

Целью данной работы является исследование навыков конвейерной обработки данных.

Для достижения поставленной цели необходимо выполнить следующие задачи:

— описать основы конвейерной обработки данных;

— описать алгоритмы, реализуемые на каждом этапе конвейера;

— разработать описанные алгоритмы;

— разработать конвейерную обработку данных;

— разработать последовательную обработку данных;

— провести тестирование реализованных алгоритмов;

— выполнить замеры процессорного времени работы конвейерной и последовательной обработок;

— провести сравнительный анализ по времени работы двух видов обработок.

1 Аналитическая часть

В данном разделе представлено теоретическое описание конвейерной обработки данных и алгоритмов каждой стадии обработки.

1.1 Конвейерная обработка данных

Конвейерная обработка данных — это подход, позволяющий создавать эффективные параллельные реализации обычных неспециализированных алгоритмов [1]. Общая идея конвейера связана с разбиением некоторого процесса обработки объектов на несколько независимых этапов и организацией одновременного (параллельного) выполнения этих этапов обработки различных объектов, передвигающихся по конвейеру от одного этапа к другому. При движении объектов по конвейеру на разных его участках выполняются разные операции, а при достижении каждым объектом конца конвейера он окажется полностью обработанным [2].

При реализации конвейерной обработки каждый из этапов должен:

1) получить данные;

2) обработать данные;

3) передать данные следующему этапу.

1.2 Стадии конвейерной обработки

Для проведения исследования в данной лабораторной работе на первой ленте конвейера будет происходить поиск обратной матрицы, на второй — умножение матриц стандартным алгоритмом, на третьей — умножение матриц алгоритмом Винограда.

1.3 Стандартный алгоритм умножения матриц

Стандартный алгоритм умножения матриц является реализацией математического определения произведения матриц, которое формулируется следующим образом: пусть даны две

$$A_{np} = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1m} \\ a_{21} & a_{22} & \dots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{l1} & a_{l2} & \dots & a_{lm} \end{pmatrix}, \tag{1.1}$$

имеющая n строк и p столбцов,

$$B_{pm} = \begin{pmatrix} b_{11} & b_{12} & \dots & b_{1n} \\ b_{21} & b_{22} & \dots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{m1} & b_{m2} & \dots & b_{mn} \end{pmatrix}, \tag{1.2}$$

имеющая p строк и m столбцов, тогда матрица C будет иметь n строк и m столбцов

$$C_{nm} = \begin{pmatrix} c_{11} & c_{12} & \dots & c_{1n} \\ c_{21} & c_{22} & \dots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{l1} & c_{l2} & \dots & c_{ln} \end{pmatrix}, \tag{1.3}$$

где

$$c_{ij} = \sum_{r=1}^{m} a_{ir} b_{rj} \quad (i = \overline{1, l}; j = \overline{1, n}) \tag{1.4}$$

- называется **произведением матриц** A и B.

1.4 Алгоритм Винограда для умножения матриц

Из результата умножения двух матриц следует, что каждый элемент в нем представляет собой скалярное произведение соответствующих строки и столбца исходных матриц. Такое умножение допускает предварительную обработку, позволяющую часть работы выполнить заранее.

Рассмотрим два вектора $V = (v_1, v_2, v_3, v_4)$ и $W = (w_1, w_2, w_3, w_4)$. Их скалярное произведение равно: $V \cdot W = v_1w_1 + v_2w_2 + v_3w_3 + v_4w_4$, что эквивалентно (1.5):

$$V \cdot W = (v_1+w_2)(v_2+w_1)+(v_3+w_4)(v_4+w_3)-v_1v_2-v_3v_4-w_1w_2-w_3w_4. \quad (1.5)$$

Несмотря на то, что второе выражение требует вычисление большего количества операций, чем стандартный алгоритм: вместо четырёх умножений – шесть, а вместо трёх сложений – десять, выражение в правой части последнего равенства допускает предварительную обработку: его части можно вычислить заранее и запомнить для каждой строки первой матрицы и для каждого столбца второй, что позволит для каждого элемента выполнять лишь два умножения и пять сложений, складывая затем только лишь с 2 предварительно посчитанными суммами соседних элементов текущих строк и столбцов. Из-за того, что операция сложения быстрее операции умножения в ЭВМ, на практике алгоритм должен работать быстрее стандартного.[3]

Вывод

В данном разделе была описана модель конвейерной обработки данных, также были представлены алгоритмы, выполняющиеся на каждой обработчике конвейера.

2 Конструкторская часть

В данном разделе будут рассмотрены алгоритмы конвейерной и последовательной обработок, а также алгоритм нахождения обратной матрицы методом Жордана-Гаусса, стандартный алгоритм умножения матриц, алгоритм Винограда для умножения матриц.

2.1 Разработка алгоритмов

На рисунке 2.1 представлена схема линейного алгоритма обработки заявок. На рисунке 2.2 представлена схема главного потока, запускающего потоки обработчиков конвейера, схемы алгоритмов которых представлены на рисунках 2.3-2.5.

На рисунках 2.6 — 2.7 представлена схема алгоритма нахождения обратной матрицы, на рисунке 2.8 — схема алгоритма умножения матриц стандартным методом, на рисунках 2.9—2.12 — схема умножения матриц алгоритмом Винограда.

Рисунок 2.1 – Схема алгоритма линейной обработки заявок

Рисунок 2.2 – Схема главного потока конвейерной обработки заявок

Рисунок 2.3 – Схема потока первого этапа обработки

Рисунок 2.4 – Схема потока второго этапа обработки

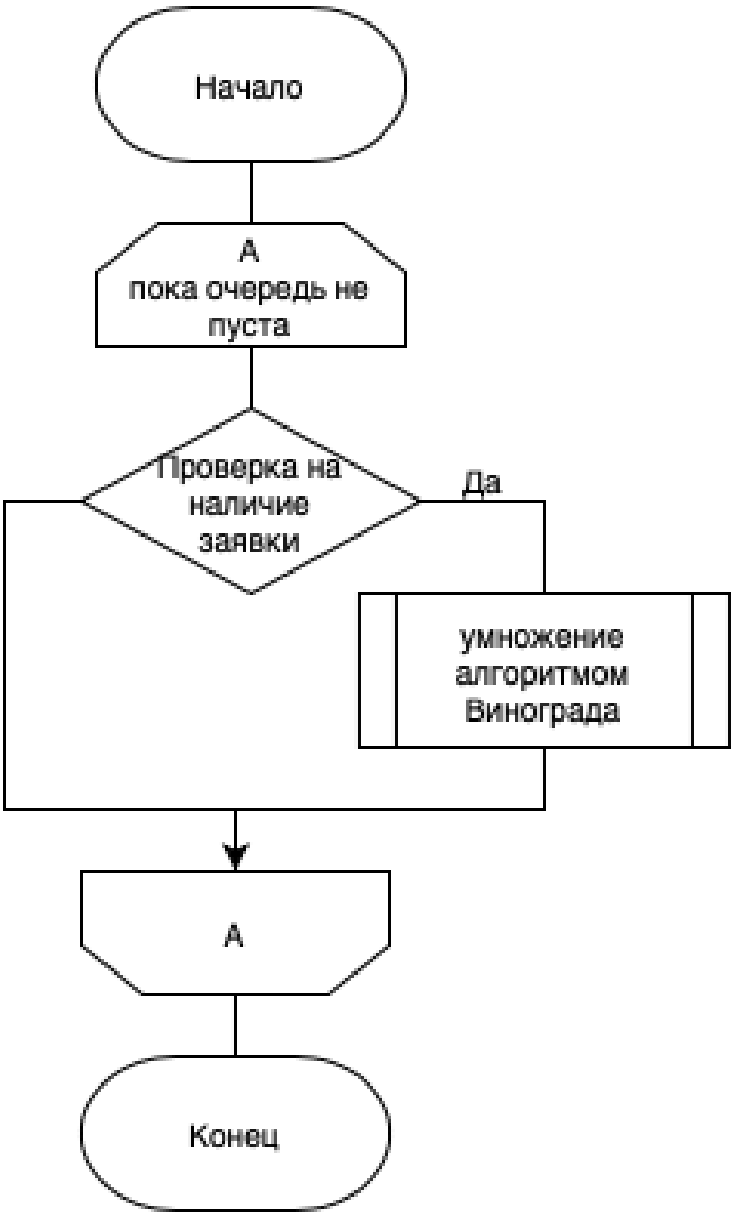

Рисунок 2.5 – Схема потока третьего этапа обработки

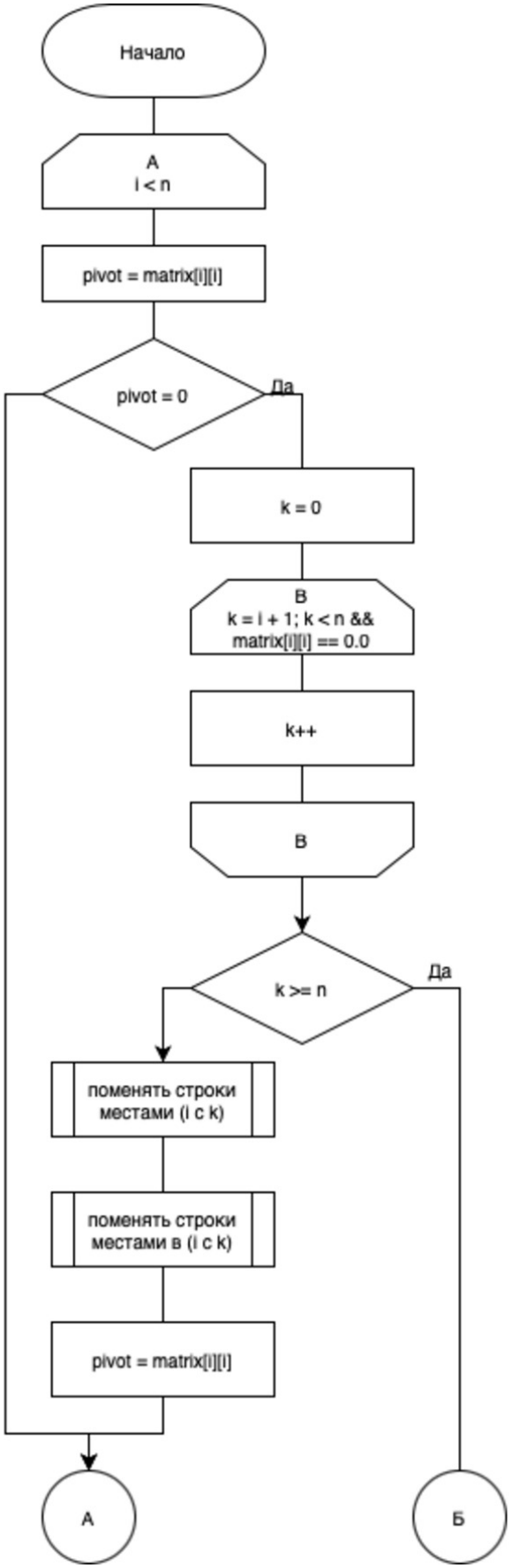

Рисунок 2.6 – Схема алгоритма поиска обратной матрицы методом Жордана-Гаусса. Часть 1

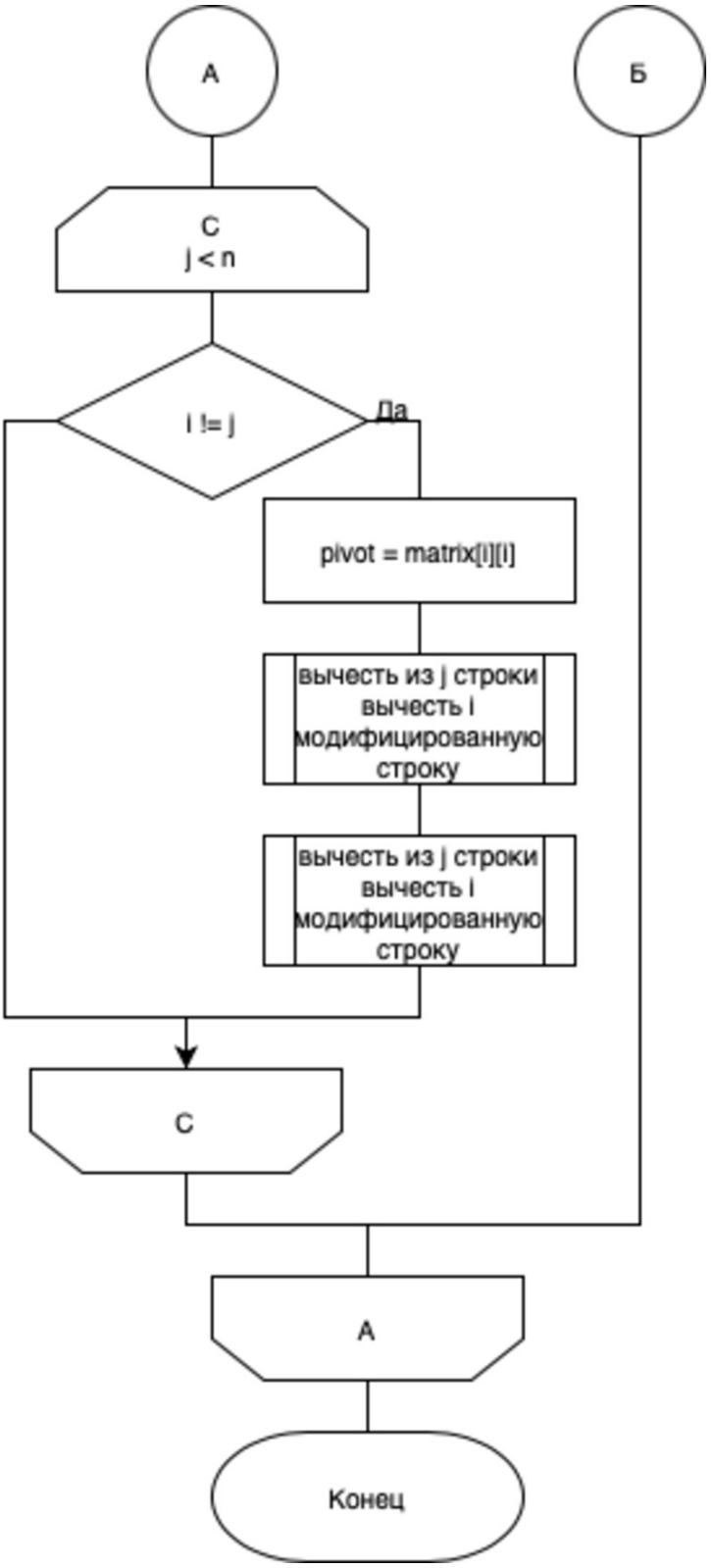

Рисунок 2.7 – Схема алгоритма поиска обратной матрицы методом Жордана-Гаусса. Часть 2

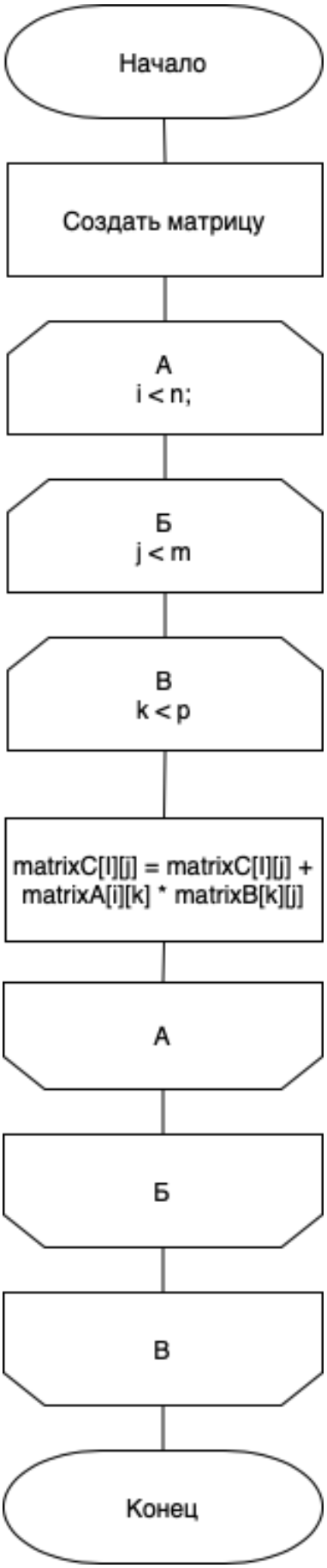

Рисунок 2.8 – Схема алгоритма умножения матриц стандартным методом

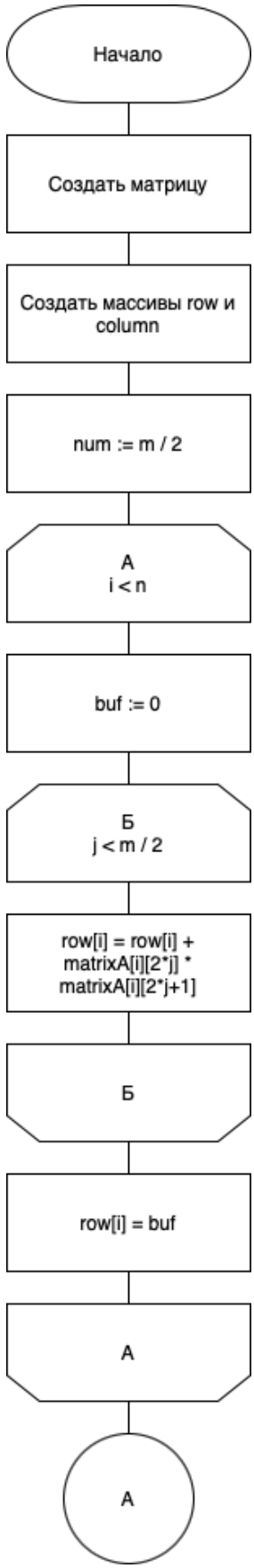

Рисунок 2.9 – Схема алгоритма Винограда. Часть 1

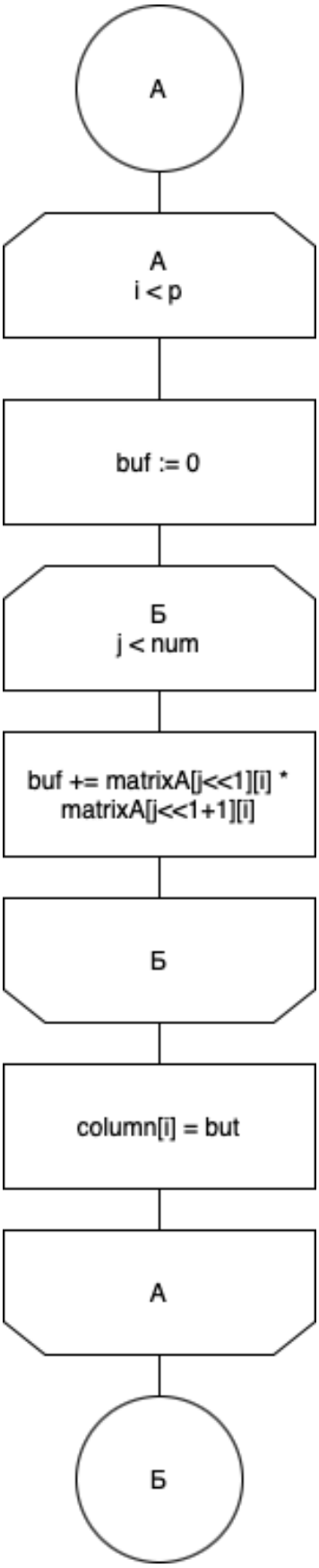

Рисунок 2.10 – Схема алгоритма Винограда. Часть 2

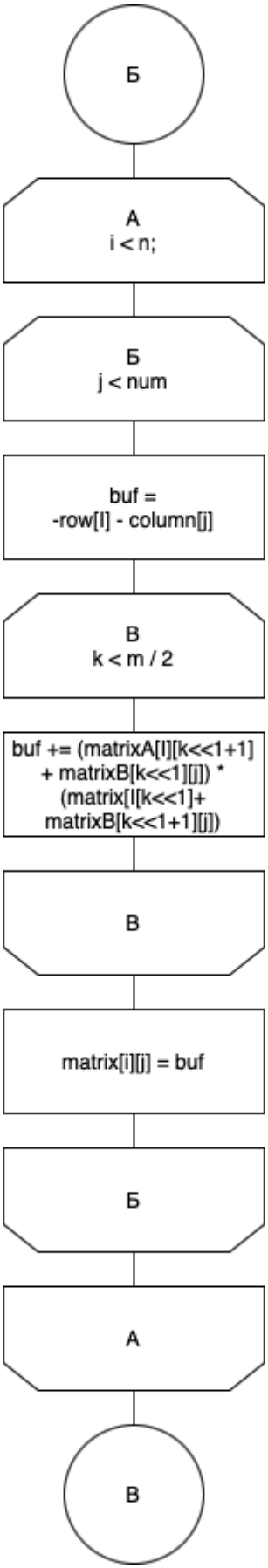

Рисунок 2.11 – Схема алгоритма Винограда. Часть 3

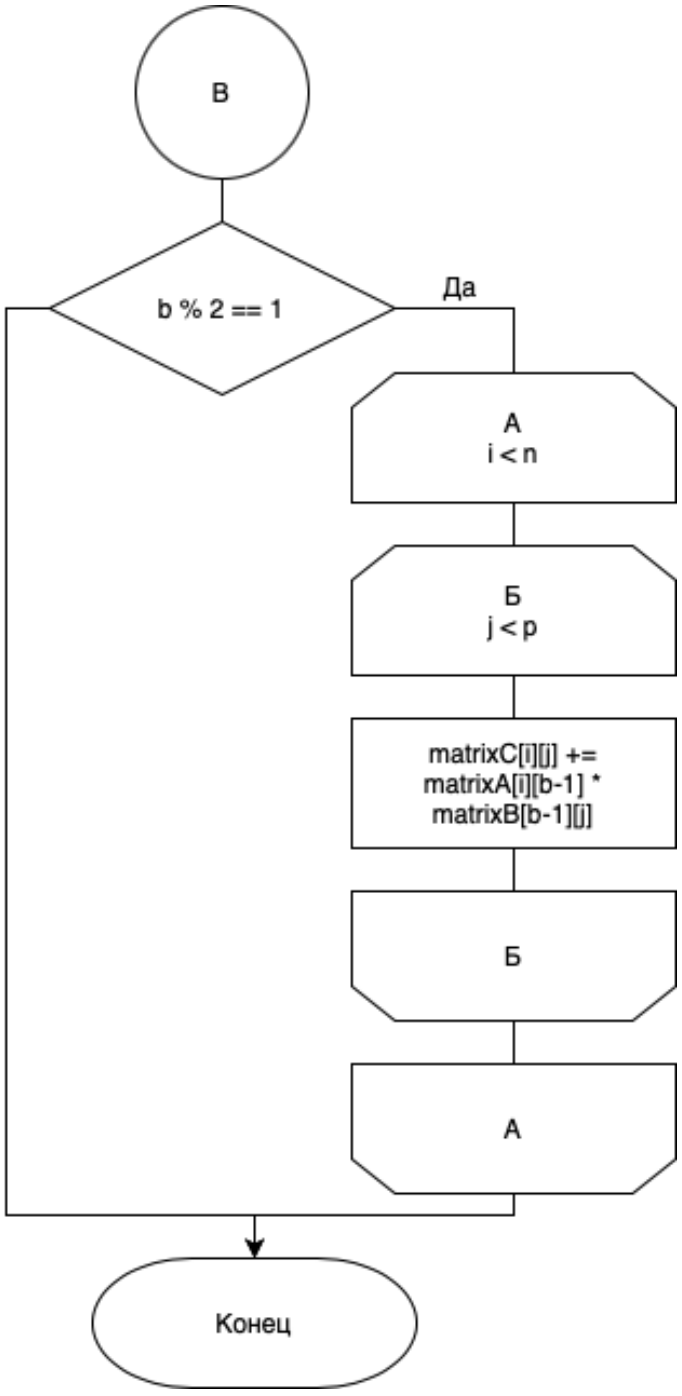

Рисунок 2.12 – Схема алгоритма Винограда. Часть 4

2.2 Структура разрабатываемого программного обеспечения

Для реализации разрабатываемого программного обеспечения будет использоваться метод структурного программирования. Каждый из алгоритмов будет представлен отдельной функцией, при необходимости будут выделены подпрограммы для каждой из них. Также будут реализованы функции для ввода-вывода и функция, вызывающая все подпрограммы для связности и полноценности программы.

2.3 Требования к программному обеспечению

Программа должна предоставлять следующие возможности:

— ввод количества обрабатываемых заявок, размерности матриц, элементов матриц;

— вывод результата работы программы;

— замер времени работы алгоритмов;

Вывод

В данном разделе были разработаны алгоритмы этапов обработки матриц, а также алгоритмы линейной и конвейерной обработки заявок, была описана структура разрабатываемого программного обеспечения.

3 Технологическая часть

В данном разделе описаны требования к программному обеспечению, средства реализации, приведены листинги кода и данные, на которых будет проводиться тестирование.

3.1 Средства реализации

Для реализации данной лабораторной работы выбран компилируемый многопоточный язык програмирования `Go` [4], так как он предоставляет необходимый функционал для работы с потоками и простоую реализацию их взаимодействия.

Замеры времени проводились при помощи getThreadCpuTimeNs функции, написанной на C, подключенной с помощью CGO [5].

3.2 Реализация алгоритмов

В данном подразделе представлены листинги кода алгоритмов:

— реализация линейной обработки 3.1;

— реализация конвейерной обработки 3.2;

— реализация стандартного алгоритма умножения матриц 3.3

— реализация алгоритма Винограда умножения матриц 3.4

Листинг 3.1 – Релизация линейной обработки

```
1 func serialMtr(sliceMtr []matrixStruct, count int) {
2     for i := range sliceMtr {
3         sliceMtr[i].inverseMatrix =
              gaussMethodEx(sliceMtr[i].matrixA)
4         sliceMtr[i].standartMul =
              mulStandart(sliceMtr[i].inverseMatrix,
              sliceMtr[i].matrixB, sliceMtr[i].size)
5         sliceMtr[i].vinogradMul =
              mulVinogradov(sliceMtr[i].inverseMatrix,
              sliceMtr[i].matrixB, sliceMtr[i].size)
6     }
7 }
```

Листинг 3.2 – Релизация линейной обработки

```
1  func pipelineMtr(sliceMtr []matrixStruct, count, flag int) {
2      inverseChan := make(chan matrixStruct, count)
3      standartChan := make(chan matrixStruct, count)
4      vinogradChan := make(chan matrixStruct, count)
5      resultChan := make(chan matrixStruct, count)
6
7      var wg sync.WaitGroup
8
9      inverse := func() {
10         defer wg.Done()
11         for {
12             select {
13             case item, ok := <-inverseChan:
```

```
14                  if ok {
15                      item.inverseMatrix =
                            gaussMethodEx(item.matrixA)
16                      standartChan <- item
17                  } else {
18                      close(standartChan)
19                      return
20                  }
21              }
22          }
23      }
24
25      standart := func() {
26          defer wg.Done()
27          for {
28              select {
29              case item, ok := <-standartChan:
30                  if ok {
31                      item.standartMul =
                            mulStandart(item.inverseMatrix,
                            item.matrixB, item.size)
32                      vinogradChan <- item
33                  } else {
34                      close(vinogradChan)
35                      return
36                  }
37              }
38          }
39      }
40
41      vinograd := func() {
42          defer wg.Done()
43          for {
44              select {
45              case item, ok := <-vinogradChan:
46                  if ok {
47                      item.vinogradMul =
                            mulVinogradov(item.inverseMatrix,
                            item.matrixB, item.size)
48                      resultChan <- item
49                  } else {
```

```
50                      close(resultChan)
51                      return
52                  }
53              }
54          }
55      }
56
57      go inverse()
58      go standart()
59      go vinograd()
60
61      wg.Add(3)
62
63      for i := 0; i < count; i++ {
64          inverseChan <- sliceMtr[i]
65      }
66
67      close(inverseChan)
68      wg.Wait()
69
70      for item := range resultChan {
71          fmt.Println(item)
72          sliceMtr[item.uniq-1] = item
73      }
74  }
```

Листинг 3.3 – Реализация алгоритма стандартного метода умножения матриц

```
1  func mulStandart(matrixA, matrixB [][]float64, n int)
       [][]float64 {
2      matrixC := make([][]float64, n)
3      for i := 0; i < n; i++ {
4          matrixC[i] = make([]float64, n)
5          for j := 0; j < n; j++ {
6              var a float64
7              for k := 0; k < n; k++ {
8                  a += matrixA[i][k] * matrixB[k][j]
9              }
10             matrixC[i][j] = a
11         }
12     }
13     return matrixC
14 }
```

Листинг 3.4 – Реализация алгоритма Винограда умножения матриц

```
1  func mulVinogradov(matrixA, matrixB [][]float64, n int)
       [][]float64 {
2      row := make([]float64, n)
3      column := make([]float64, n)
4      matrixC := make([][]float64, n)
5
6      num := n / 2
7
8      for i := 0; i < n; i++ {
9          matrixC[i] = make([]float64, n)
10         buf := 0.0
11         for j := 0; j < num ; j++ {
12             buf += matrixA[i][j << 1] * matrixA[i][j << 1 + 1]
13         }
14         row[i] = buf
15     }
16
17     for i := 0; i < n; i++ {
18         buf := 0.0
19         for j := 0; j < num; j++ {
20             buf += matrixB[j << 1][i] * matrixB[j << 1 + 1][i]
21         }
```

```
22          column[i] = buf
23      }
24
25      for i := 0; i < n; i++ {
26          for j := 0; j < n; j++ {
27              buf := -row[i] - column[j]
28              for k := 0; k < num; k++ {
29                  buf += (matrixA[i][k << 1 + 1] + matrixB[k <<
                        1][j]) *
30                      (matrixA[i][k << 1] + matrixB[k << 1 +
                            1][j])
31              }
32              matrixC[i][j] = buf
33          }
34      }
35
36      if (n % 2 == 1) {
37          l := n - 1
38          for i := 0; i < n; i++ {
39              for j := 0; j < n; j++ {
40                  matrixC[i][j] += matrixA[i][l] * matrixB[l][j]
41              }
42          }
43      }
44
45      return matrixC
46  }
```

3.3 Функциональные тесты

В таблице 3.1 представлены функциональные тесты для алгоритмов умножения матриц, а в таблице 3.2 приведены функциональные тесты для алгоритма нахождения обратной матрицы.

Таблица 3.1 – Функциональные тесты

Матрица 1	**Матрица 2**	**Ожидаемый результат**
$\begin{pmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \\ 1 & 2 & 3 \end{pmatrix}$	$\begin{pmatrix} -1 & 2 & 3 \\ 1 & -2 & 3 \\ 1 & 2 & -3 \end{pmatrix}$	$\begin{pmatrix} 4 & 4 & 0 \\ 4 & 4 & 0 \\ 4 & 4 & 0 \end{pmatrix}$
$\begin{pmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \end{pmatrix}$	$\begin{pmatrix} -1 & -3 \\ -2 & -2 \\ -3 & -1 \end{pmatrix}$	$\begin{pmatrix} -14 & -10 \\ -14 & -10 \end{pmatrix}$
$\begin{pmatrix} 3 & -5 \\ 1 & -2 \end{pmatrix}$	$\begin{pmatrix} 2 & -5 \\ 1 & -3 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$
$\begin{pmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \\ 1 & 2 & 3 \end{pmatrix}$	$\begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$	$\begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$
$\begin{pmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \\ 1 & 2 & 3 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \\ 1 & 2 & 3 \end{pmatrix}$

Таблица 3.2 – Функциональные тесты

Матрица	**Ожидаемый результат**
$\begin{pmatrix} 5 & 9 & 2 \\ 6 & 11 & 4 \\ 1 & 3 & 1 \end{pmatrix}$	$\begin{pmatrix} 0.1 & 0.3 & -2 \\ 0.2 & 0 & 0.9 \\ -1 & 0.7 & 0 \end{pmatrix}$
$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 3 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 0.5 & 0 \\ 0 & 0 & 0.33333 \end{pmatrix}$
$\begin{pmatrix} 2 & 1 & -1 \\ 3 & 2 & -2 \\ 1 & -1 & 2 \end{pmatrix}$	$\begin{pmatrix} 2 & -1 & 0 \\ -8 & 5 & 1 \\ -5 & 3 & 1 \end{pmatrix}$
$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$	$\begin{pmatrix} -2 & 1 \\ 1.5 & -0.5 \end{pmatrix}$

Алгоритмы, реализованные в данной лабораторной работе, функциональные тесты прошли успешно.

Вывод

В данном разделе были реализованы последовательный и конвейрный алгоритмы поэтапной обрабокти матриц. Также были написаны функциональные тесты для проверки правильности работы программы.

4 Исследовательская часть

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялись замеры по времени, представлены далее:

— Процессор – 2 ГЦ 4-ядерный процессор Intel Core i5;

— Оперативная память – 16 ГБайт;

— Операционная система – macOS Venura 13.5.2.

4.2 Пример работы

В данном подразделе представлен пример 4.1 работы программы:

Листинг 4.1 – Пример работы программы

```
ivanmamvriyskiy@MacBook-Pro-Ivan-2 main % go run *.go
1)Запустить конвейер
2)Последовательная обработка
3)Замер времени
0)Выход
1
Введите количество заявок:1
Введите размерность матрицы:3

Ввод матрицы A
Введите 1 элемент 1 строки: 1
Введите 2 элемент 1 строки: 2
Введите 3 элемент 1 строки: 3
Введите 1 элемент 2 строки: 4
Введите 2 элемент 2 строки: 0
Введите 3 элемент 2 строки: 5
Введите 1 элемент 3 строки: 6
Введите 2 элемент 3 строки: 7
Введите 3 элемент 3 строки: 0

Ввод матрицы B
```

```
Введите 1 элемент 1 строки: 1
Введите 2 элемент 1 строки: 2
Введите 3 элемент 1 строки: 3
Введите 1 элемент 2 строки: 4
Введите 2 элемент 2 строки: 5
Введите 3 элемент 2 строки: 6
Введите 1 элемент 3 строки: 7
Введите 2 элемент 3 строки: 8
Введите 3 элемент 3 строки: 9

Обратная матрица:
[-0.3211009174311927 0.1926605504587156 0.09174311926605505]
[-2.2018348623853212 1.3211009174311927 -0.5137614678899083]
[-3.5 -0.625 1]
Стандартный алгоритм умножения:
[1.0917431192660552 1.0550458715596331 1.0183486238532113]
[-0.5137614678899083 -1.908256880733945 -3.3027522935779823]
[1 -2.125 -5.25]
Алгоритм Винограа:
[1.091743119266055 1.0550458715596323 1.0183486238532105]
[-0.5137614678899078 -1.908256880733946 -3.302752293577983]
[1 -2.125 -5.25]
```

4.3 Время выполнения алгоритмов

В таблице 4.1 представлены замеры времени конвейерной и последовательной обработок в зависимости от размерности матриы.

На рисунке 4.1 представлен граф зависимости времени от размерности матрицы.

Таблица 4.1 – Время выполнения работы алгоритмов в зависимости от размерности матрицы (мс)

Размерность	Конвейер	Последовательная
1	9	5
10	20	35
50	101	210
100	200	406
200	454	909
300	546	1 239
400	722	1 671
500	955	1 848
600	1 235	2 219
700	1 299	2 879
800	1 386	3 126
900	1 549	3 261
1000	1 656	3 659

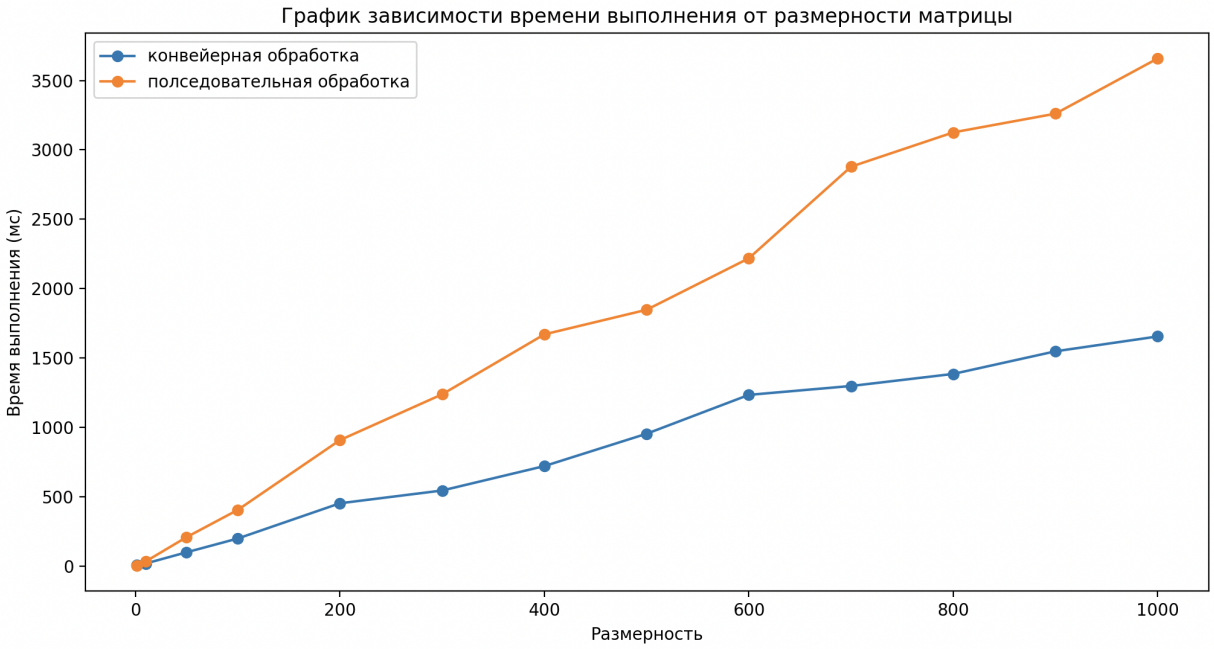

Рисунок 4.1 – График зависимости времени выполнения работы от размерности матрицы

4.4 Вывод

Таким образом, для ускорения обработки объектов, которая состоит из нескольких этапов, необходимо использовать конвейерную обработку.

Заключение

В ходе исследования был проведен сравнительный аналализ последовательной и конвейерной реализации алгоритма поэтапной обработки матриц, в результате которого была выяснено, что конвейерна обработка работает прмиерно в 1.5 – 2.5 раза быстрее.

Исходя из данных показателей для более быстрой обработки заявок необходимо использовать конвейер.

Цель данной лабораторной работы, которая заключается в исследовании конвейерной обработки достигнута, также были выполнены следующие задачи:

— описаны основы конвейерной обработки данных;

— описаны алгоритмы, реализуемые на каждом этапе конвейера;

— разработана конвейерная обработка данных;

— разработаны алгоритмы;

— разработана последовательная обработка данных;

— проведено тестирование реализованных алгоритмов;

— выполнены замеры процессорного времени работы конвейерной и последовательной обработок;

— проведен сравнительный анализ по времени работы двух видов обработок.

Список используемых источников

1. Применение конвейерной обработки данных на примере сортировки простыми вставками / Д. А. Погорелов, А. М. Таразанов, Л. Л. Волкова [и др.] // Образование и наука в России и за рубежом. 2019. Т. 49, № 1.

2. Богословский Н. Н. Конспект лекций дисциплины «Архитектура и программное обеспечение высокопроизводительных вычислительных систем». Режим доступа: `http://hpc-education.ru/files/lectures/2011/bogoslovskiy/bogoslovskiy_2011_lectures05.pdf` (дата обращения: 10.12.2023).

3. Макконелл Дж. Основы современных алгоритмов. 2-е доп. изд. М.: Техносфера, 2004. с. 368.

4. The Go Programming Language [Электронный ресурс]. Режим доступа: `https://golang.org/` (дата обращения: 10.12.2023).

5. Документация CGo [Электронный ресурс]. Режим доступа: `https://pkg.go.dev/cmd/cgo` (дата обращения: 10.12.2023).