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»

Лабораторная работа № 2
по дисциплине «Анализ Алгоритмов»

Тема Алгоритмы умножения матриц

Студент Пермякова Е. Д.

Группа ИУ7-52Б

Преподаватели Строганов Д. В., Волкова Л. Л

Москва, 2024

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Целью работы является выполнение оценки ресурсной эффективности алгоритмов умножения матриц и их реализации.

Задачи:

1) Описать математическую основу стандартного алгоритма и алгоритма Винограда умножения матриц;
2) Описать модель вычислений;
3) Разобрать алгоритм умножения матриц – стандартный, Винограда, оптимизированный согласно варианту алгоритм Винограда;
4) Выполнить оценку трудоемкости разработанных алгоритмов либо их реализации;
5) Реализовать разработанные алгоритмы в програмном обеспечении с 2 режимами работы – одиночного расчета и массированного замера процессорного времени выполнения реализации каждого алгоритма;
6) Выполнить замеры процессорного времени выполнения реализации разработанных алгоритмов в зависимости от варьируемого размера матриц;
7) Выполнить сравнительный анализ рассчитанных трудоемкости и результатов замера процессорного времени выполнения реализации трех алгоритмов с учетом лучшего и худшего случаев по трудоемкости;

1 Аналитическая часть

В данной работе будут рассмотрены алгоритмы умножения матриц: стандартный и Винограда.

1.1 Стандартный алгоритм умножения матриц

Пусть даны две матрицы Пусть даны две матрицы A и B размерности $l \times m$ и $m \times n$ соответственно:

$$A_{lm} = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1m} \\ a_{21} & a_{22} & \dots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{l1} & a_{l2} & \dots & a_{lm} \end{pmatrix}, \quad B_{mn} = \begin{pmatrix} b_{11} & b_{12} & \dots & b_{1n} \\ b_{21} & b_{22} & \dots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{m1} & b_{m2} & \dots & b_{mn} \end{pmatrix}, \tag{1.1}$$

тогда матрица C размерностью $l \times n$

$$C_{ln} = \begin{pmatrix} c_{11} & c_{12} & \dots & c_{1n} \\ c_{21} & c_{22} & \dots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{l1} & c_{l2} & \dots & c_{ln} \end{pmatrix}, \tag{1.2}$$

где

$$c_{ij} = \sum_{r=1}^{m} a_{ir} b_{rj} \quad (i = \overline{1, l}; j = \overline{1, n}) \tag{1.3}$$

будет называться произведением матриц A и B.

Операция умножения двух матриц выполнима только в том случае, если число столбцов в первом сомножителе равно числу строк во втором.

1.2 Алгоритм Винограда

Ключевая идея алгоритма Винограда – снизить долю операций умножения, заменив их операциями сложения, которые являются более эффек-

тивными по времени.

Пусть есть 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 (1.4):

$$V \cdot W = v_1 w_1 + v_2 w_2 + v_3 w_3 + v_4 w_4 \tag{1.4}$$

что эквивалентно (1.5):

$$\begin{aligned} V \cdot W = (v_1 + w_2)(v_2 + w_1) + (v_3 + w_4)(v_4 + w_3) - \\ - v_1 v_2 - v_3 v_4 - w_1 w_2 - w_3 w_4. \end{aligned} \tag{1.5}$$

Выражения $-v_1 v_2 - v_3 v_4$ и $-w_1 w_2 - w_3 w_4$ можно вычислить заранее и использовать повторно при умножении строки V матрицы A на все столбцы W матрицы B.

Это позволит выполнить меньшее количество операций умножения: 2 умножения и 5 сложений, вместо 4 умножений и 4 сложений. Но при нечетном значении размера матрицы нужно дополнительно добавить произведения крайних элементов соответствующих строк и столбцов.

Операция сложения выполняется быстрее, поэтому на практике алгоритм должен работать быстрее обычного алгоритма перемножения матриц.

Вывод

В данном разделе были теоретически разобраны два алгоритмы умножения матриц: стандартного и Винограда.

2 Конструкторская часть

В этом разделе будут представлens схемы алгоритмов умножения матриц: стандартного, Винограда и оптимизированного алгоритма Винограда.

2.1 Описание алгоритмов

На рисунках 2.1-2.5 представлены схемы алгоритмов перемножения матриц.

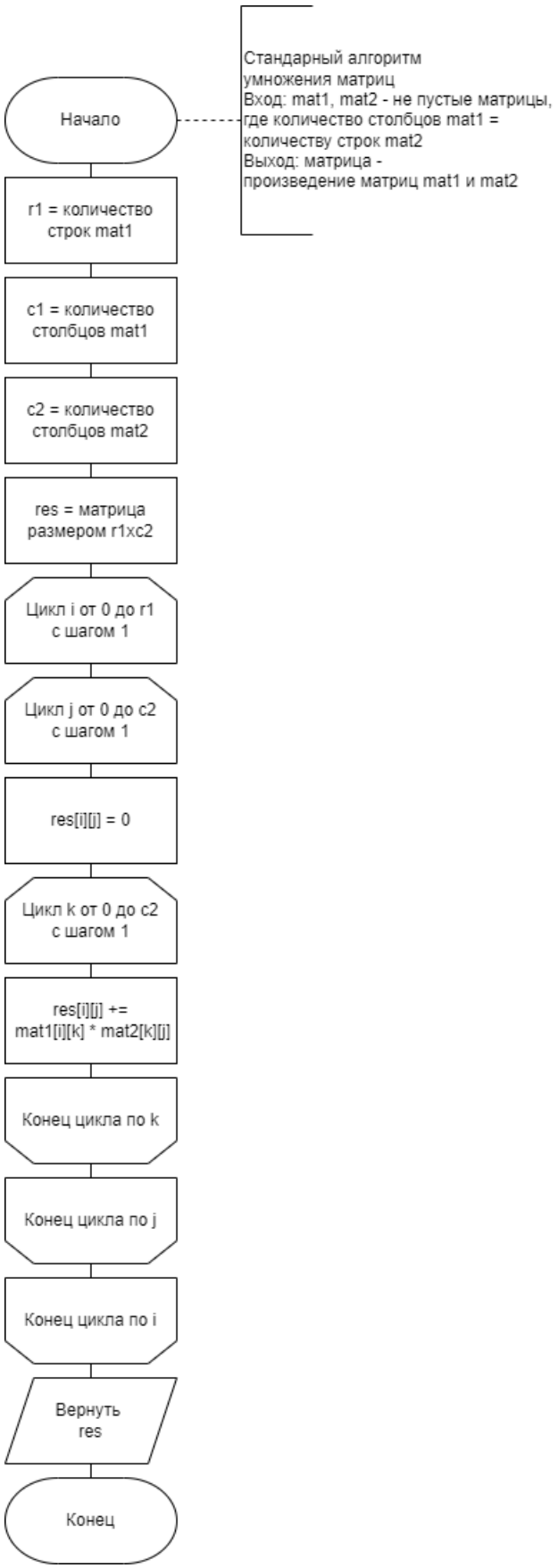

Рисунок 2.1 — Схема стандартного алгоритма умножения матриц

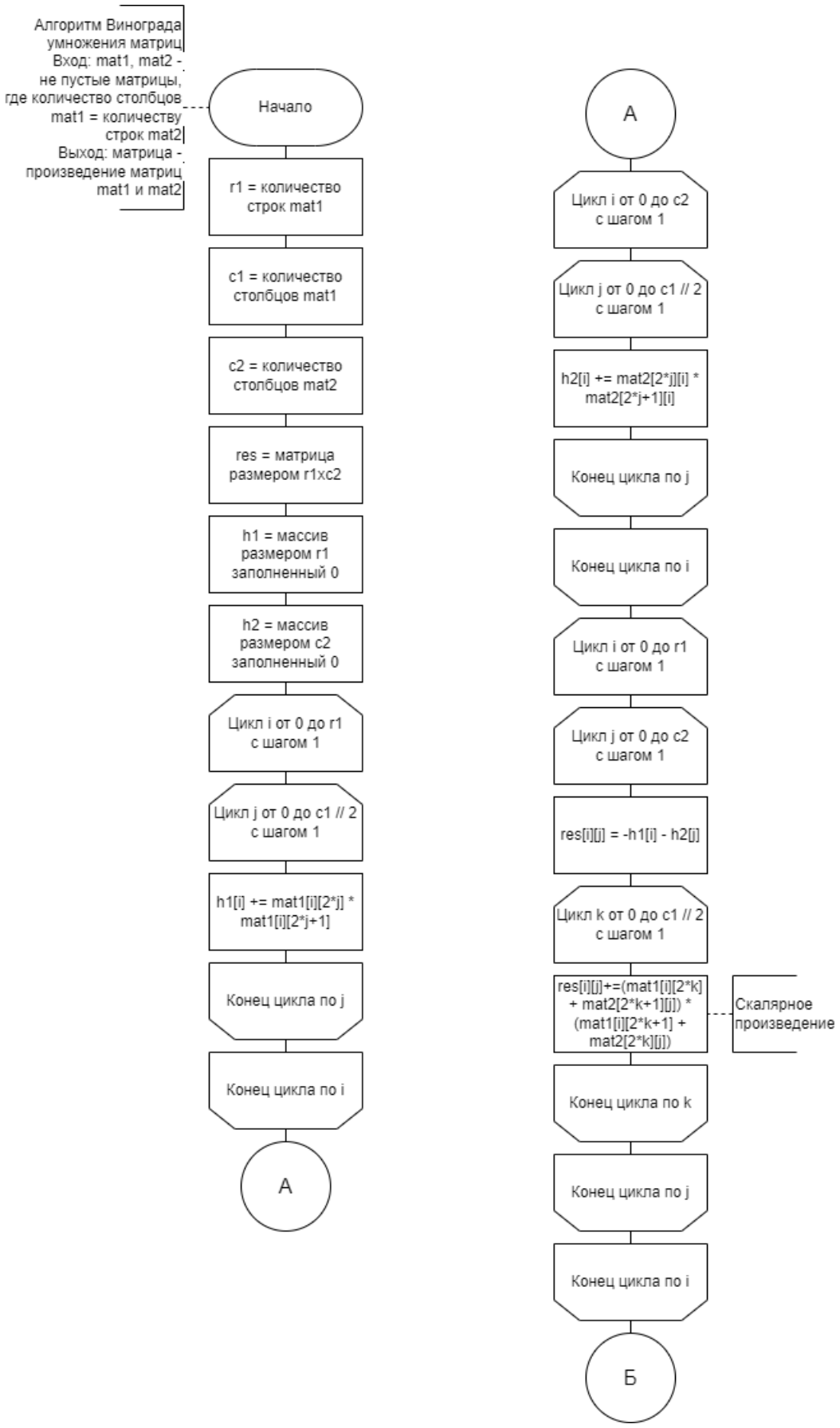

Рисунок 2.2 — Схема алгоритма Винограда

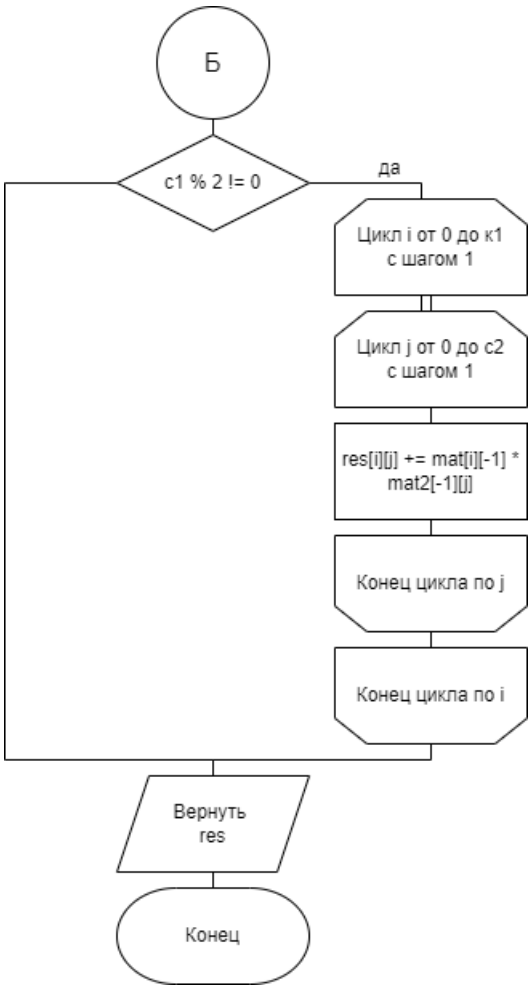

Рисунок 2.3 — Схема алгоритма Винограда

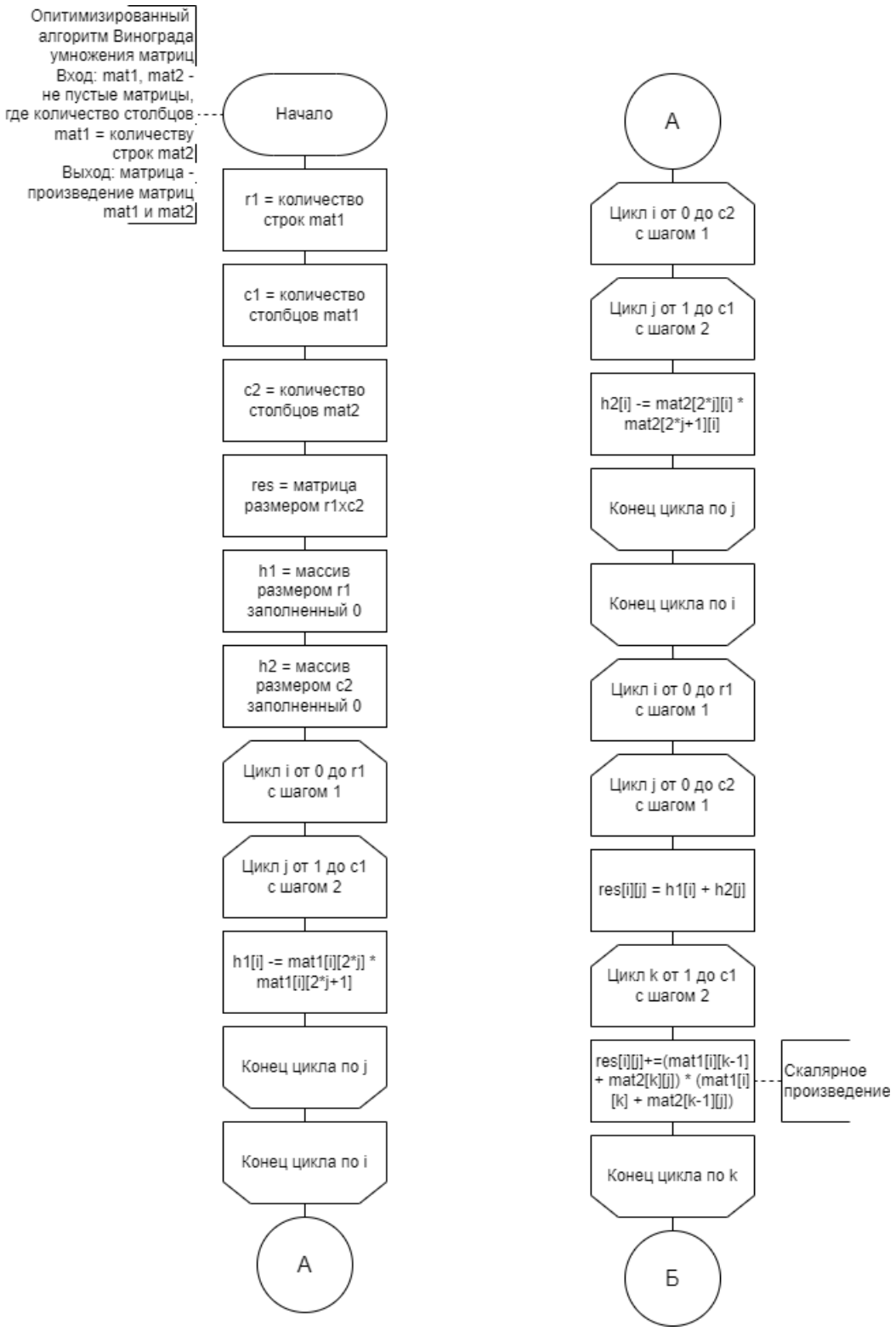

Рисунок 2.4 — Схема оптимизированного алгоритма Винограда

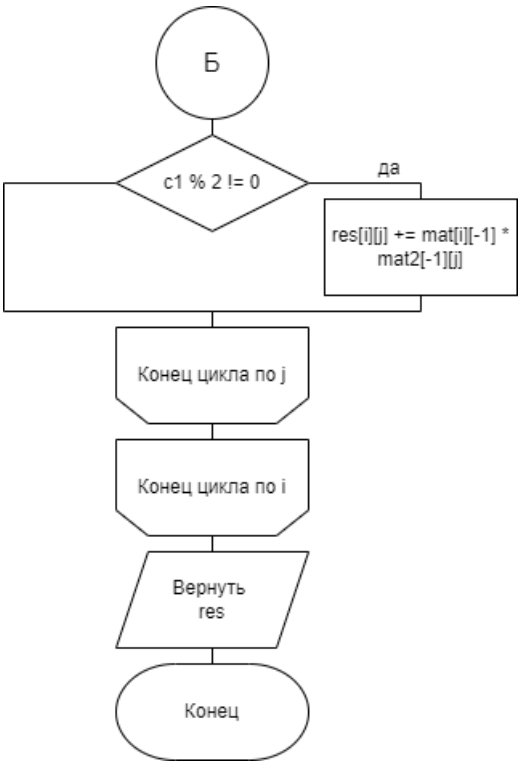

Рисунок 2.5 — Схема оптимизированного алгоритма Винограда

Вывод

В данном разделе были представлены схемы алгоритмов умножения матриц: стандартного, Винограда и оптимизированного алгоритма Винограда.

3 Технологическая часть

В данном разделе будут приведены средства реализации, листинг кода, функциональные тесты, модель вычислений и трудоемкость алгоритмов.

3.1 Средства реализации

В данной работе для реализации был выбран язык программирования $Python$ [1], так как он удовлетворяет требованиям лабраторной работы: поддерживает динамические структуры данных, такие как массивы и имеет библиотеку $Matplotlib$ [2] для построения графиков.

3.2 Реализация алгоритмов

В листингах 3.1-3.3 представлены реализации алгоритмов поиска элемента в массиве.

Листинг 3.1 – Стандартный алгоритм умножения матриц

```
        def standard_matrix_mul(mat1, rows1, cols1, mat2,
           rows2, cols2, res):
            for i in range(rows1):
                for j in range(cols2):
                res[i][j] = 0
                for k in range(cols1):
                    res[i][j] += mat1[i][k] * mat2[k][j]
```

Листинг 3.2 – Алгоритм Винограда

```
    def vinograd_matrix_mul(mat1, rows1, cols1, mat2, rows2,
       cols2, res):
        help_mat1 = [0] * rows1
        help_mat2 = [0] * cols2

        for i in range(rows1):
            for j in range(cols1 // 2):
                help_mat1[i] += mat1[i][2 * j] * mat1[i][2 * j
                   + 1]
```

```
        for i in range(cols2):
            for j in range(cols1 // 2):
                help_mat2[i] += mat2[2 * j][i] * mat2[2 * j +
                    1][i]

        for i in range(rows1):
            for j in range(cols2):
            res[i][j] = -help_mat1[i] - help_mat2[j]
            for k in range(cols1 // 2):
                res[i][j] += (mat1[i][2 * k] + mat2[2 * k +
                    1][j]) * (mat1[i][2 * k + 1] + mat2[2 *
                    k][j])

        if cols1 % 2 != 0:
            for i in range(rows1):
                for j in range(cols2):
                    res[i][j] += mat1[i][-1] * mat2[-1][j]
```

Листинг 3.3 – Оптимизированный алгоритм Винограда

```
    def new_vinograd_matrix_mul(mat1, rows1, cols1, mat2,
        rows2, cols2, res):
        help_mat1 = [0] * rows1
        help_mat2 = [0] * cols2

        for i in range(rows1):
            for j in range(1, cols1, 2):
                help_mat1[i] -= mat1[i][j - 1] * mat1[i][j]

        for i in range(cols2):
            for j in range(1, cols1, 2):
                help_mat2[i] -= mat2[j - 1][i] * mat2[j][i]

        for i in range(rows1):
            for j in range(cols2):
                res[i][j] = help_mat1[i] + help_mat2[j]
                for k in range(1, cols1, 2):
                    res[i][j] += (mat1[i][k - 1] + mat2[k][j])
                        * (mat1[i][k] + mat2[k - 1][j])
                if cols1 % 2 != 0:
                    res[i][j] += mat1[i][-1] * mat2[-1][j]
```

3.3 Модель вычислений

Для вычисления трудоемкости алгоритмов была введена модель вычислений:

1) Операции из списка (3.1) имеют трудоемкость 1:

$$=, +, -, + =, - =, ==, ! =, <, <=, >, >=, [], <<, >>, and, or \quad (3.1)$$

2) Операции из списка (3.2) имеют трудоемкость 2:

$$*, /, //, \%, * =, / =, // =, * = \quad (3.2)$$

3) Пусть трудоемкость условного перехода = 0, тогда трудоемкость условного оператора вида `if условие then блок1 else блок2` рассчитывается, как (3.3):

$$f_{if} = f_{\text{условия}} + \begin{cases} min(f_1, f_2), & \text{л. с. (лучший случай),} \\ max(f_1, f_2), & \text{х. с. (худший случай).} \end{cases} \quad (3.3)$$

4) Трудоемкость цикла с N шагами рассчитывается, как (3.4);

$$f_{for} = f_{\text{инициализации}} + f_{\text{сравнения}} + N(f_{\text{инкремента}} + f_{\text{сравнения}} + f_{\text{тела}}) \quad (3.4)$$

5) трудоемкость вызова функции равна 0.

3.4 Трудоемкость алгоритмов

Трудоемкость реализованных алгоритмов умножения матриц:

3.4.1 Стандартный алгоритм умножения матриц

Для стандартного алгоритма умножения матриц размером $M \times N$ и $N \times Q$ трудоемкость состоит из:

— внешнего цикла по $i \in [1..M]$, трудоемкость которого: $f = 2 + M \cdot (2 + f)$;

— цикла по $j \in [1..Q]$, трудоемкость которого: $f = 2 + Q \cdot (2 + f)$;
— инициализации элемента результирующей матрицы в строке i и столбце j нулем: $f = 3$;
— цикла по $k \in [1..N]$, трудоемкость которого: $f = 2 + 14N$.

Поскольку трудоемкость стандартного алгоритма равна трудоемкости внешнего цикла, то:

$$f_{standard} = 2 + M \cdot (4 + Q \cdot (2 + 3 + 2 + N(2 + 12))); \tag{3.5}$$

$$f_{standard} = 14MNQ + 7MQ + 4M + 2 \approx 14MNQ; \tag{3.6}$$

3.4.2 Алгоритм Винограда

Для алгоритма Винограда трудоемкость состоит из:
— Заполнения вспомагательного массива a_tmp, трудоемкость которого (3.7):

$$f_{a\_tmp} = 2 + M(2 + 4 + \frac{N}{2}(4 + 4 + 11)) = \frac{19}{2}MN + 6M + 2; \tag{3.7}$$

— Заполнения вспомагательного массива b_tmp, трудоемкость которого (3.8):

$$f_{b\_tmp} = 2 + Q(2 + 4 + \frac{N}{2}(4 + 4 + 11)) = \frac{19}{2}QN + 6Q + 2; \tag{3.8}$$

— Цикла заполнения результирующей матрицы, трудоемкость которого (3.10):

$$f_c = 2 + M(2 + 2 + Q(2 + 7 + 4 + \frac{N}{2}(4 + 6 + 22))); \tag{3.9}$$

$$f_c = \frac{32}{2}MNQ + 13MQ + 4M + 2; \tag{3.10}$$

— Дополнительного цикла в случае если N не четная, трудоемкость

которого (3.12):

$$f_{last} = 3+ \\ + \begin{cases} 0, & \text{л. с.,} \\ 2 + M(2 + 2 + Q(2 + 11)), & \text{х. с. при N \% 2 == 1} \end{cases} \tag{3.11}$$

$$f_{last} = 3 + \begin{cases} 0, & \text{л. с.,} \\ 13MQ + 4M + 2, & \text{х. с. при N \% 2 == 1} \end{cases} \tag{3.12}$$

Тогда трудоемкость алгоритма Винограда составит (3.14):

$$f_{vin} = f_{a\_tmp} + f_{b\_tmp} + f_c + f_{last} \tag{3.13}$$

$$\begin{aligned} f_{vin} = {} & 16MNQ + \frac{19}{2}QN + \frac{19}{2}MN + \\ & + 10M + 6Q + 13MQ + 6 + \\ & + \begin{cases} 0, & \text{л. с.,} \\ 13MQ + 4M + 2, & \text{х. с. при N \% 2 == 1} \end{cases} \end{aligned} \tag{3.14}$$

3.4.3 Оптимизированный алгоритм Винограда

Оптимизация заключается в:

— инкремент счётчика наиболее вложенного цикла на 2;

— объединение III и IV частей алгоритма Винограда;

— введение декремента при вычислении вспомогательных массивов;

Тогда трудоемкость оптимизированного алгоритма Винограда состоит из:

— Заполнения массива a_tmp (3.15):

$$f_{a\_tmp} = 2 + M(2 + 2 + \frac{N}{2}(2 + 9)) = \frac{11}{2}MN + 4M + 2 \tag{3.15}$$

— Заполнения массива b_tmp (3.16):

$$f_{b\_tmp} = 2 + Q(2 + 2 + \frac{N}{2}(2 + 9)) = \frac{11}{2}QN + 4M + 2 \tag{3.16}$$

— Цикла заполнения результирующей матрицы с учетом случая когда N нечетная, трудоемкость которого (3.18):

$$f_c = 2 + M(2 + 2 + Q(2 + 6 + 2 + \frac{N}{2}(2 + 20) + \\ + 3 + \begin{cases} 0, & \text{л. с.,} \\ 12, & \text{х. с. при N \% 2 == 1} \end{cases}); \tag{3.17}$$

$$f_c = 11MNQ + 13MQ + 4M + 2 + \\ + MQ \begin{cases} 0, & \text{л. с.,} \\ 12, & \text{х. с. при N \% 2 == 1} \end{cases} \tag{3.18}$$

Тогда трудоемкость оптимизированного алгоритма Винограда составит (3.19):

$$f_{new_vin} = 11MNQ + 13MQ + \frac{11}{2}QN + \\ \frac{11}{2}MN + 12M + 6 + MQ \begin{cases} 0, & \text{л. с.,} \\ 12, & \text{х. с. при N \% 2 == 1} \end{cases} \tag{3.19}$$

3.5 Классы эквивалентности тестирования

Для тестирования были выделены следующие классы тестирования:

1) Пустые матрицы;
2) Матрицы размером 1x1;
3) Матрицы размером 3x2 и 2x3;
4) Матрицы размером 3x3 и 3x3;

3.6 Функциональные тесты

В таблице 3.1 приведены функциональные тесты для алгоритмов умножения матриц.

Все тесты пройдены успешно.

Таблица 3.1 — Функциональные тесты

Матрица 1	Матрица 2	Ожидаемый результат
$\begin{pmatrix} \end{pmatrix}$	$\begin{pmatrix} \end{pmatrix}$	$\begin{pmatrix} \end{pmatrix}$
$\begin{pmatrix} 1 \end{pmatrix}$	$\begin{pmatrix} 2 \end{pmatrix}$	$\begin{pmatrix} 2 \end{pmatrix}$
$\begin{pmatrix} 2 & 3 \\ 4 & 5 \\ 6 & 7 \end{pmatrix}$	$\begin{pmatrix} 10 & 11 & 12 \\ 20 & 21 & 23 \end{pmatrix}$	$\begin{pmatrix} 80 & 85 & 93 \\ 140 & 149 & 163 \\ 200 & 213 & 233 \end{pmatrix}$
$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$

Вывод

Были представлены средства реализации, листинг кода, функциональные тесты, модель вычислений и трудоемкость алгоритмов.

По результатам вычисления трудоемкости алгоритмов оптимизированный алгоритм Винограда должен работать быстрее остальных, а неоптимизированный алгоритм Винограда медленнее. ($f_{new_vin} < f_{std} < f_{vin}$, $11MNQ < 14MNQ < 16MNQ$)

4 Исследовательская часть

Цель исследования – сравнительный анализ реализованных алгоритмов по трудоемкости.

4.1 Технические характеристики

— Операционная система – Майкрософт Windows 11 Домашняя для одного языка; Версия – 10.0.22631; Сборка – 22631;
— Установленная оперативная память (RAM) – 16,0 ГБ;
— Процессор – AMD Ryzen 7 5800H with Radeon Graphics, 3201 МГц, ядер: 8, логических процессоров: 16;
— Микроконтроллер – STM32F303 [3];

4.2 Время выполнения алгоритмов

Замеры времени работы алгоритмов проводились на плате и для этого использовалась функция ticks_ms(...) из библиотеки time на MicroPython [4].

Замеры проводились для четных размеров матриц от 2 до 40 по 10 раз на различных входных матрицах. А также – для нечетных размеров матриц от 3 до 39 по 10 раз на различных входных данных.

Были полученны графики зависимости времени работы плгоритма от размеров квадратных матриц 4.1-4.2 для алгоритмов умножения матриц.

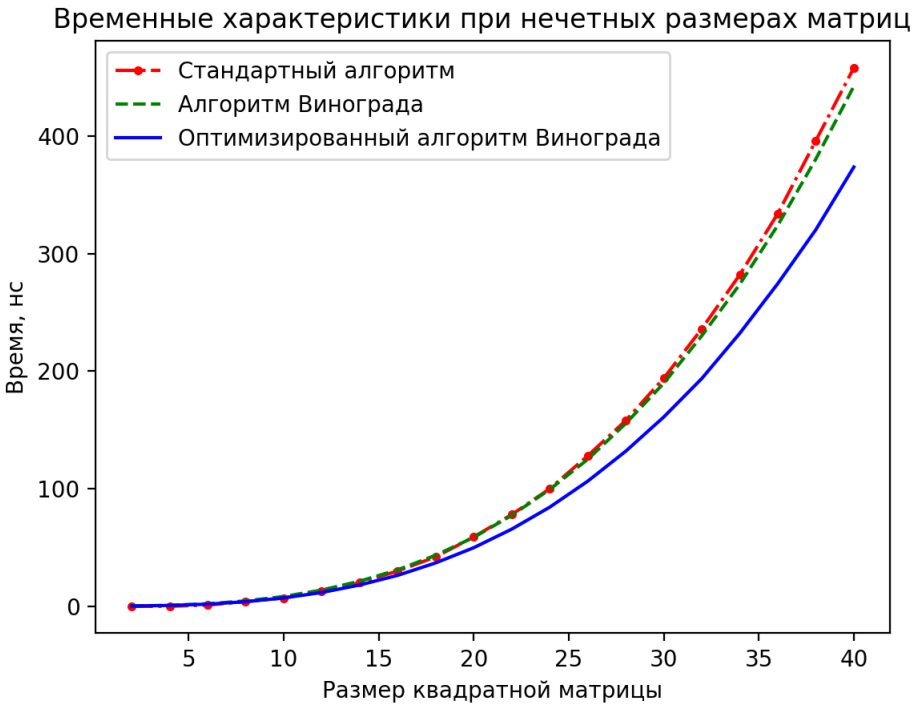

Рисунок 4.1 — Сравнение времени работы алгоритмов умножения матриц нечетного размера

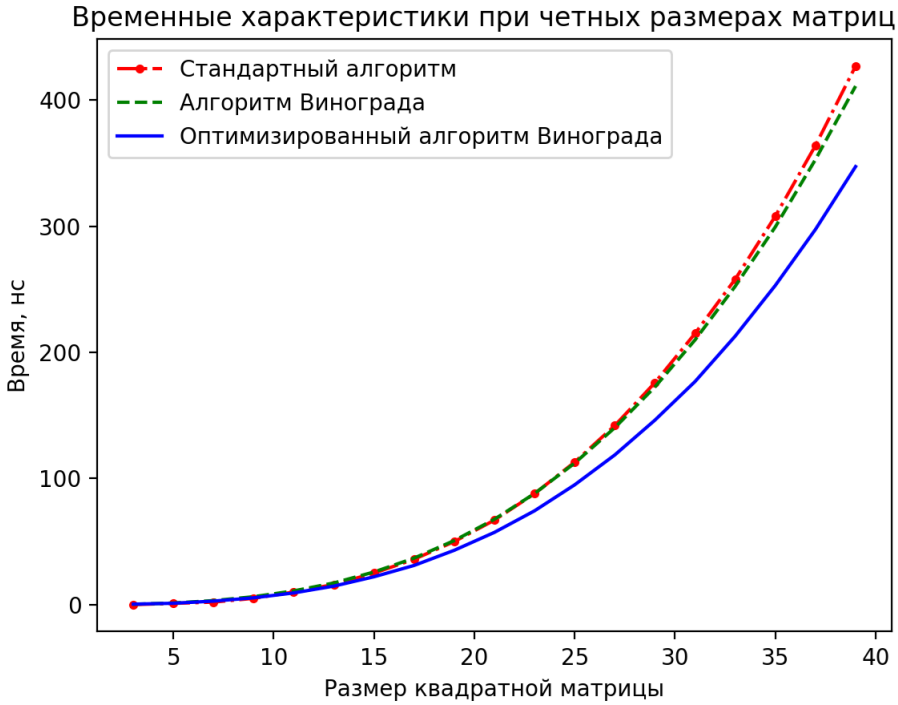

Рисунок 4.2 — Сравнение времени работы алгоритмов матриц четного размера

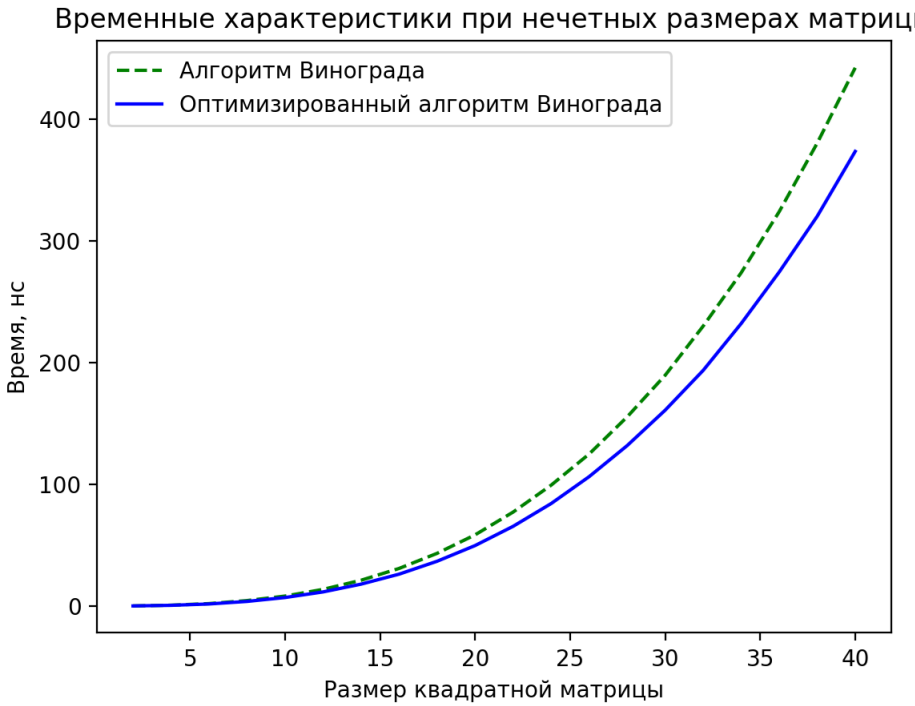

Рисунок 4.3 — Сравнение времени работы алгоритмов умножения матриц нечетного размера для разных реализаций алгоритма винограда

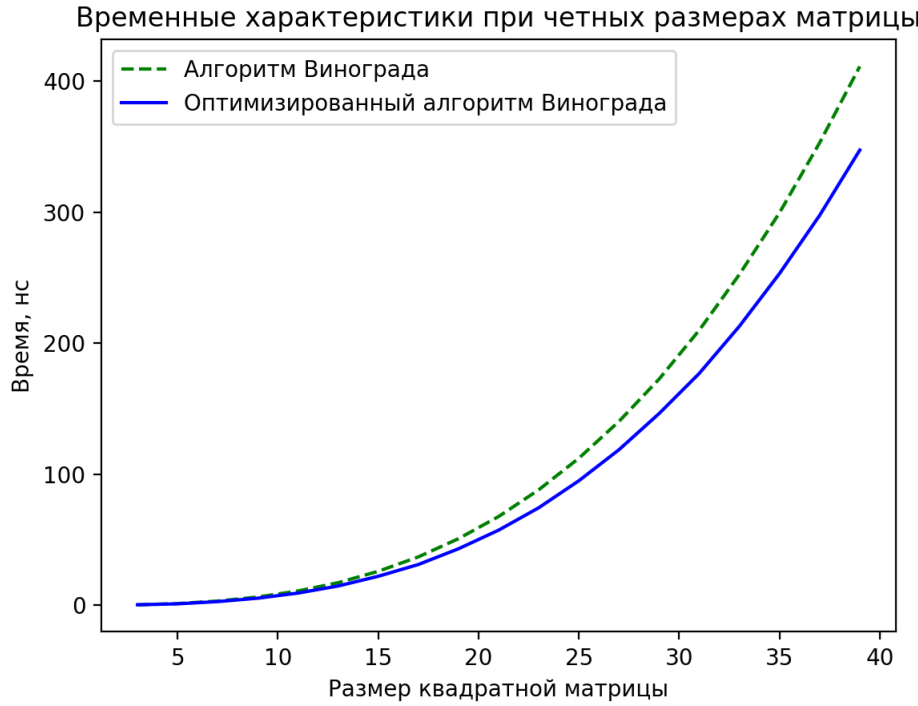

Рисунок 4.4 — Сравнение времени работы алгоритмов матриц четного размера для разных реализаций алгоритма винограда

4.3 Вывод

Из проведённых замеров можно сделать следующие выводы:

— Оптимизированный алгоритм Винограда демонстрирует наилучшие результаты по времени работы на всех тестовых данных, как в расчитанной ранее формула, так и на практике;

— Как и ожидалось умножение матрицы нечетного размера требует больше времени;

— На практике стандартный алгоритм умножения матриц работает медленнее, чем неоптимизированный алгоритм Винограда, это можно обьяснять оптимизацией компилятора или неточность введенной модели вычислений.

ЗАКЛЮЧЕНИЕ

В ходе лабораторной работы была выполнена поставленнная цель, которая заключалась в выполнении оценки ресурсной эффективности алгоритмов умножения матриц и их реализации.

Были выполнены следующие задачи:

1) Описать математическую основу стандартного алгоритма и алгоритма Винограда умножения матриц;
2) Описать модель вычислений;
3) Разобрать алгоритм умножения матриц – стандартный, Винограда, оптимизированный согласно варианту алгоритм Винограда;
4) Выполнить оценку трудоемкости разработанных алгоритмов либо их реализации;
5) Реализовать разработанные алгоритмы в програмном обеспечении с 2 режимами работы – одиночного расчета и массированного замера процессорного времени выполнения реализации каждого алгоритма;
6) Выполнить замеры процессорного времени выполнения реализации разработанных алгоритмов в зависимости от варьируемого размера матриц;
7) Выполнить сравнительный анализ рассчитанных трудоемкости и результатов замера процессорного времени выполнения реализации трех алгоритмов с учетом лучшего и худшего случаев по трудоемкости;

Основываясь на проведенном исследовании можно сделать следующие выводы.

— Оптимизированный алгоритм Винограда демонстрирует наилучшие результаты по времени работы на всех тестовых данных, как в расчитанной ранее формула, так и на практике;
— Как и ожидалось умножение матрицы нечетного размера требует больше времени;
— На практике стандартный алгоритм умножения матриц работает медленнее, чем неоптимизированный алгоритм Винограда, это можно обьяснять оптимизацией компилятора или неточность введенной модели вычислений;

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Welcome to Python [Электронный ресурс]. Режим доступа: https://www.python.org. (дата обращения: 30.09.2024).

2. Matplotlib 3.9.2 documentation [Электронный ресурс]. Режим доступа: https://matplotlib.org/stable/index.html. (дата обращения: 30.09.2024).

3. ST Nucleo F767ZI — NuttX latest documentation [Электронный ресурс]. Режим доступа: https://nuttx.apache.org/docs/latest/platforms/arm/stm32f7/boards/nucleo-f767/index.html. (дата обращения: 6.10.2024).

4. time – time related functions — MicroPython latest documentation [Электронный ресурс]. Режим доступа: https://docs.micropython.org/en/latest/library/time.html. (дата обращения: 6.10.2024).