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»

НАПРАВЛЕНИЕ ПОДГОТОВКИ «09.03.04 Программная инженерия»

ОТЧЕТ

ПО ЛАБОРАТОРНОЙ РАБОТЕ №2

Название: Алгоритм Винограда

Дисциплина: Анализ алгоритмов

Студент	ИУ7-56Б		Ковель А.Д.
	Группа	Подпись, дата	И. О. Фамилия
Преподаватель			Волкова Л.Л.
		Подпись, дата	И. О. Фамилия
Преподаватель			Строганов Ю.В.
		Подпись, дата	И. О. Фамилия

Москва, 2022 г.

Оглавление

Введение

Цель лабораторной работы — изучение и анализ алгоритмов умножения матриц.

Задачи данной лабораторной следующее:

1) изучение алгоритмов перемножения матриц, измерение трудоемкости различных алгоритмов умножения матриц, применение оптимизации при реализации алгоритма умножения матриц Винограда;

2) проведение сравнительного анализа трудоемкости алгоритмов умножения матриц на основе теоретических вычислений;

3) получение экспериментального подтверждения различий по временной эффективности алгоритмов умножения матрица, путем измерения процессорного время с помощью разработанного программного обеспечения;

4) описание и обоснование полученных результатов в отчете о выполненной лабораторной работе, выполненного как расчетно-пояснительная записка к работе.

1 Аналитический раздел

В этом разделе будут представлены описания алгоритмов умножения матриц и алгоритм Винограда.

1.1 Применение математического подхода

Даны матрицы, $A \in \mathbb{R}^{m \times n}$, $B \in \mathbb{R}^{n \times p}$, произведение матриц, $C = A \times B$, каждый элемент которой вычисляется согласно формуле

$$c_{i,j} = \sum_{n}^{k=1} a_{i,k} \cdot b_{k,j}, \text{ где } i = \overline{1,m}, j = \overline{1,p} \tag{1.1}$$

Стандартный алгоритм умножения матриц реализует формулу (1.1).

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

1.2 Алгоритм Винограда

Алгоритм Винограда [1] — алгоритм умножения матриц. Рассмотрим два вектора $U = (u_1, u_2, u_3, u_4)$, $W = (w_1, w_2, w_3, w_4)$. Их скалярное произведение равно $U \cdot W = u1w1 + u2w2 + u3w3 + u4w4$, что эквивалентно:

$$U \cdot W = (u_1 + w_2)(w_1 + u_2) + (u_4 + w_3)(w_4 + u_3) - u_1u_2 - u_3u_4 - w_1w_2 - w_3w_4$$

За счет предварительной обработки последних 4 слагаемых можно получить прирост производительности.

Стоит упомянуть, что при нечетном количестве столбцов матриц нужно дополнительно добавить произведение последних элементов соответствующих строки и столбца к скалярному произведению строки и столбца.

Вывод

Была выявлена основная особенность подхода Винограда — идея предварительной обработки данных. Разница во времени выполнения реализаций этих двух алгоритмов будет экспериментально вычислена в исследовательском разделе.

2 Конструкторский раздел

В данном разделе представлены схемы алгоритмов умножений матриц и их модификации.

2.1 Трудоемкость алгоритмов

Для получения функции трудоемкости алгоритма необходимо ввести модель оценки трудоемкости. Трудоемкость "элементарных"операций оценивается следующим образом.

1. Трудоемкость 1 имеют операции:

$$+, -, =, <, >, <=, >=, ==, + =, - =,$$
$$++, --, [], \&\&, ||, >>, <<$$

2. Трудоемкость 2 имеют операции:

$$*, /, \backslash, \%$$

3. Трудоемкость конструкции ветвления определяется как

$$f_{if} = f_{\text{условие}} + \begin{cases} min\left(f_{\text{истина}}, f_{\text{ложь}}\right) & \text{в лучшем случае,} \\ max\left(f_{\text{истина}}, f_{\text{ложь}}\right) & \text{в худшем случае.} \end{cases} \tag{2.1}$$

4. Трудоемкость цикла расчитывается как

$$f_{\text{цикл}} = f_{\text{инициал.}} + f_{\text{сравн.}} + N\left(f_{\text{тело}} + f_{\text{инкремент}} + f_{\text{сравн.}}\right) \tag{2.2}$$

5. Трудоемкость вызова функции равна 0.

2.1.1 Классический алгоритм

Пусть на вход алгоритму поступают матрицы M_{left} и M_{right} с размерностью $n \times m$ и $m \times q$. Тогда трудоемкость классического алгоритма равна

$$f_{alg} = 2 + n\left(2 + f_j\right) \approx 14mnq = 14MNQ \tag{2.3}$$

2.1.2 Алгоритм Винограда

Трудоёмкость алгоритма Винограда состоит из следующих этапов:

- создания и инициализации массивов MH и MV, трудоёмкость которого равна
$$f = M + N; \tag{2.4}$$

- заполнения массива MH, трудоёмкость заполнения равна
$$f_{MH} = \frac{19}{2}MN + 6M + 2; \tag{2.5}$$

- заполнения массива MV, трудоёмкость заполнения которого равна
$$f_{MV} = \frac{19}{2}QN + 6Q + 2); \tag{2.6}$$

- цикла заполнения матрицы для чётного N, его трудоёмкость равна
$$f_{\text{цикл}} = 16MQN + 13MQ + 4M + 2; \tag{2.7}$$

- цикла для дополнения умножения суммой последних элементов перемножаемых строки и столбца, если линейная размерность N нечётная, трудоемкость этого цикла равна
$$f_{\text{последний}} = 3 + \begin{cases} 0, & \text{чётная,} \\ 16MQ + 4M + 2, & \text{иначе.} \end{cases} \tag{2.8}$$

Итого, для худшего случая (нечётный размер матриц N) трудоемкость равна
$$f = f_{MH} + f_{MV} + f_{cycle} + f_{last} \approx 16 \cdot MNQ \tag{2.9}$$

Для лучшего случая (чётный размер матриц N) трудоемкость равна

$$f = f_{MH} + f_{MV} + f_{cycle} + f_{last} \approx 16 \cdot MNQ \tag{2.10}$$

2.1.3 Оптимизированный алгоритм Винограда

Оптимизированный алгоритм Винограда представляет собой обычный алгоритм Винограда с добавлением следующих оптимизаций:

- вычисление ряда величин происходит заранее;
- используется побитовый сдвиг, вместо деления на 2;
- используется побитовый сдвиг, вместо умножения на 2.

Трудоёмкость улучшенного алгоритма Винограда состоит из:

- создания и инициализации массивов MH и MV, трудоёмкость которого равна
$$f = M + N; \tag{2.11}$$
- заполнения массива MH, трудоёмкость которого равна
$$f_{MH} = \frac{13}{2}MN + 4M + 5; \tag{2.12}$$
- заполнения массива MV, трудоёмкость которого равна
$$f_{MV} = \frac{13}{2}QN + 4Q + 5; \tag{2.13}$$
- цикла заполнения для чётных размеров, трудоёмкость которого равна
$$f_{\text{цикл}} = 2 + M \cdot (4 + N \cdot (11 + \frac{Q}{2} \cdot 21)); \tag{2.14}$$
- условие, для дополнения умножения суммой последних нечётных строки и столбца, если общий размер нечётный, трудоемкость которого равна
$$f_{\text{последний}} = 3 + \begin{cases} 0, & \text{чётная,} \\ 13MQ + 4M + 2, & \text{иначе.} \end{cases} \tag{2.15}$$

Итого, для худшего случая (нечётный размер матриц N) трудоемкость равна

$$f = f_{MH} + f_{MV} + f_{\text{цикл}} + f_{\text{последний}} \approx 10.5MNK \tag{2.16}$$

Для лучшего случая (чётный размер матриц N) трудоемкость равна

$$f = f_{MH} + f_{MV} + f_{\text{цикл}} + f_{\text{последний}} \approx 10.5MNK \tag{2.17}$$

2.2 Разработка алгоритмов

На рисунке 2.1 приведена схема классического алгоритма умножения матриц. На рисунке 2.2 приведена схема алгоритма Винограда. Рисунок 2.3 демонстрируют схему реализации оптимизированного алгоритма Винограда.

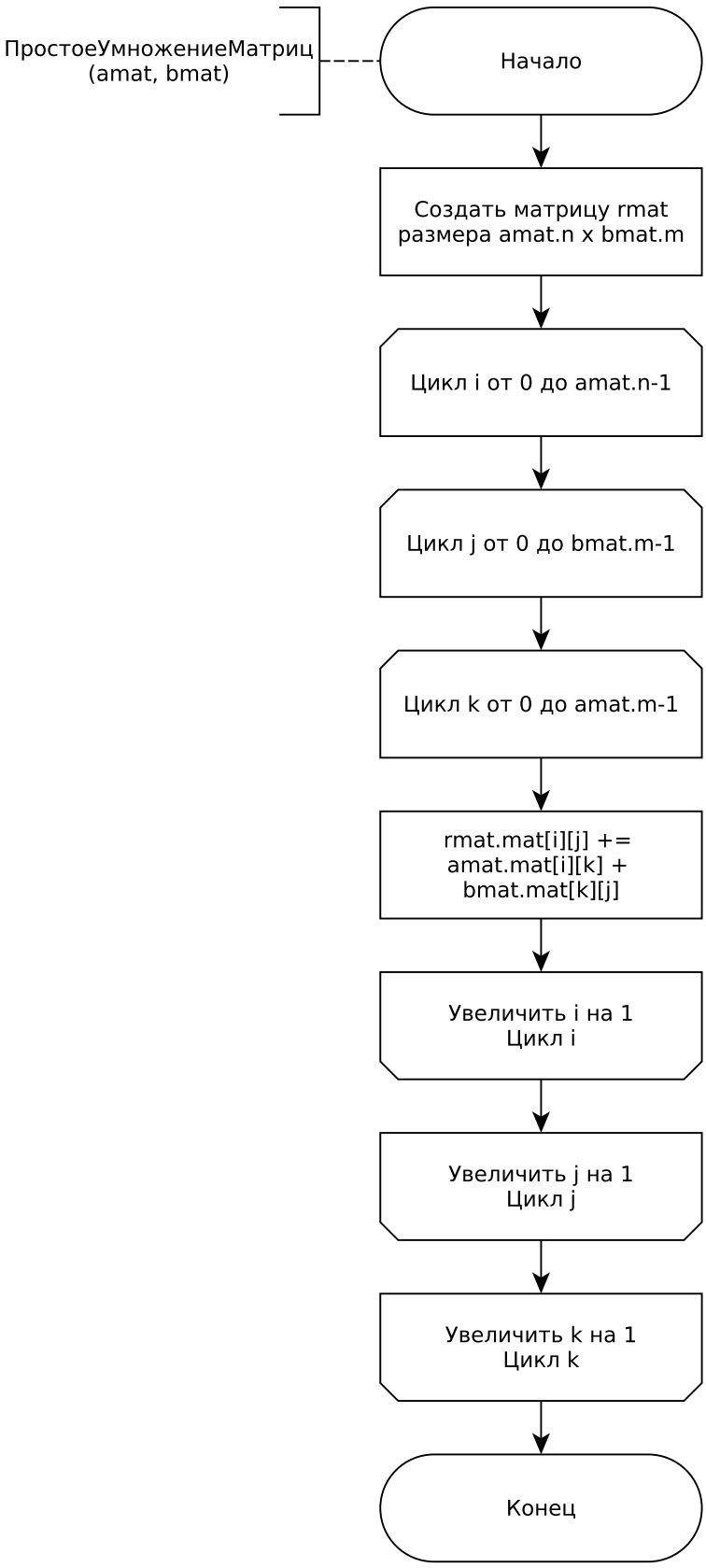

Рисунок 2.1 – Схема классического алгоритма умножения матриц

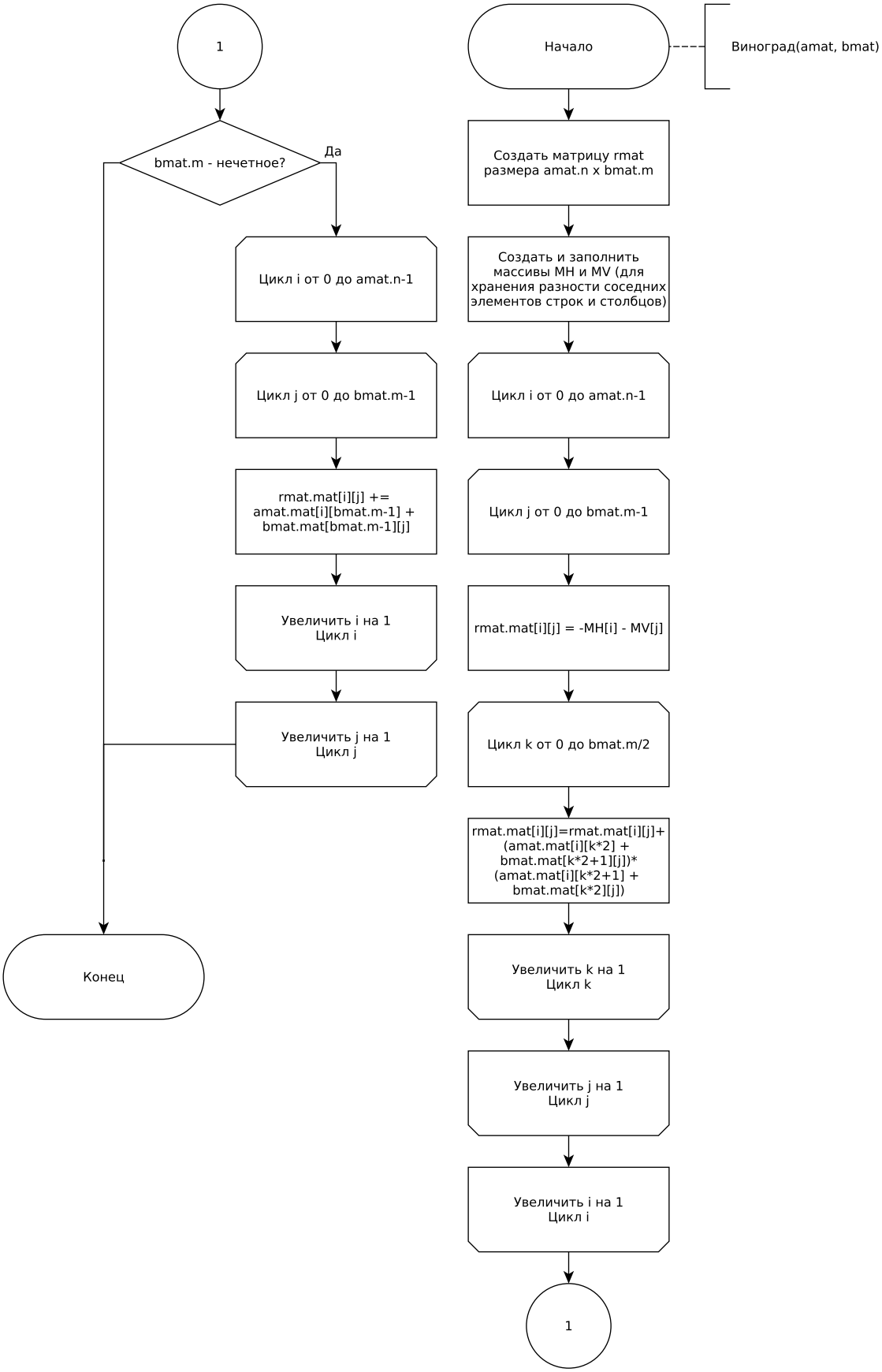

Рисунок 2.2 – Схема алгоритма умножения матриц Винограда

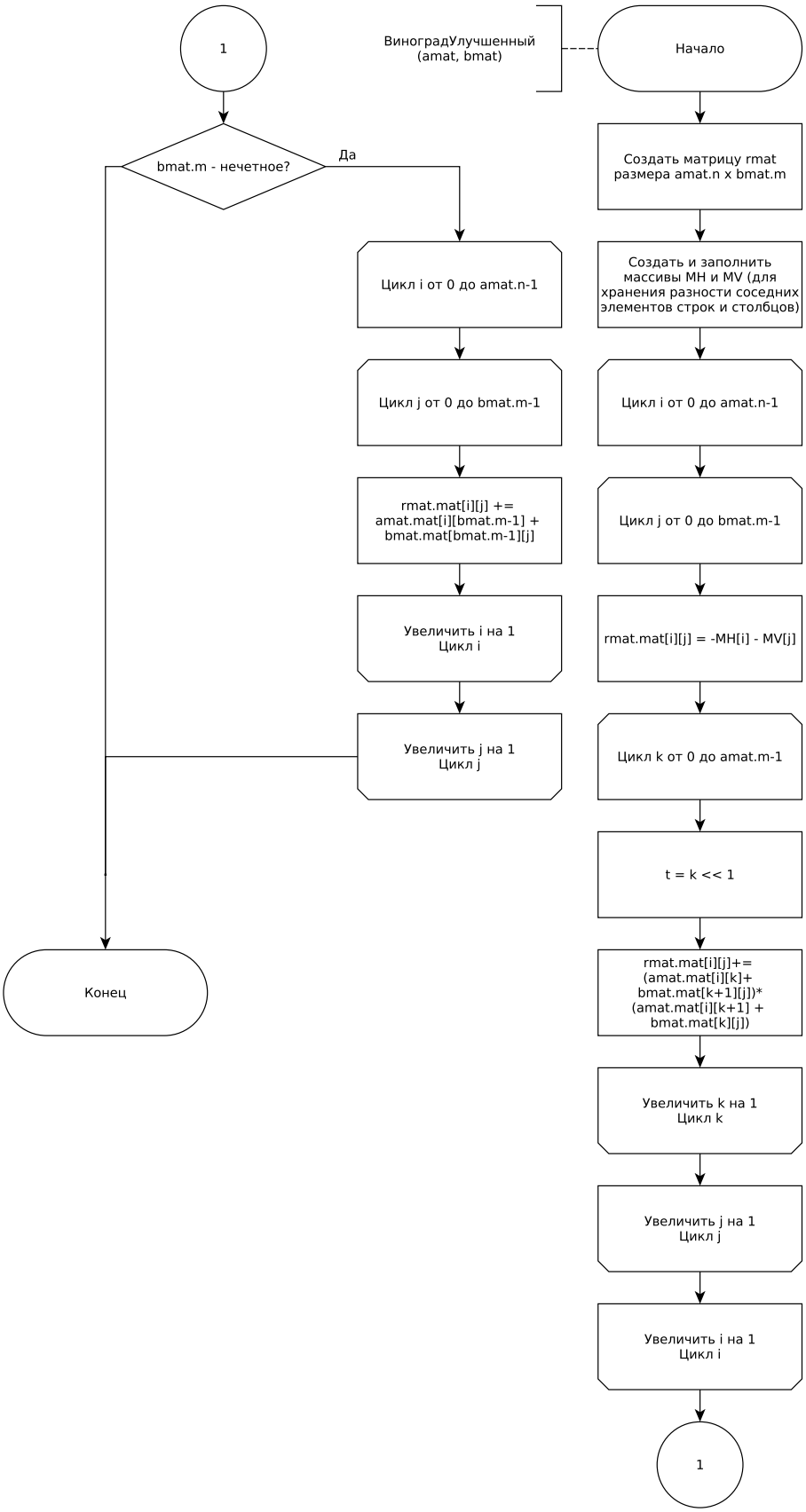

Рисунок 2.3 – Схема оптимизированного алгоритма умножения матриц Винограда

Вывод

Алгоритмы были проанализированы с точки зрения временных затрат. Было выявлено, что оптимизированный алгоритм Винограда имеет трудоемкость в 1.5 раза меньше, чем классический алгоритм Винограда.

Были построены схемы алгоритмов. Были выделены способы оптимизации алгоритма Винограда. Было получено достаточно теоретических сведений для разработки ПО, решающего поставленную задачу.

3 Технологический раздел

В данном разделе будут приведены требования к программному обеспечению, средства реализации и листинга кода.

3.1 Требования к ПО

Программное обеспечение должно удовлетворять следующим требованиям:

- программа получает на вход с клавиатуры две матрицы размеров в пределах 10000×10000 либо получает два числа — размерности матрицы в пределах 10000;
- программа выдает матрицу — произведение двух полученных матриц;
- в программе возможно измерение процессорного времени.

3.2 Средства реализации

Для реализации ПО был выбран язык программирования Python [2].

В данном языке есть все требующиеся инструменты для данной лабораторной работы.

В качестве среды разработки была выбрана среда VS Code [3], запуск происходил через команду python main.py.

3.3 Средства замера времени

Замеры времени выполнения реализаций алгоритма будут проводиться при помощи функции process_time [4] библиотеки time. Данная команда возвращает значения процессорного времени типа int в наносекундах.

Замеры времени для каждой реализации алгоритма и для каждого комплекта входных данных проводились 100 раз.

Листинг 3.1 – Пример замера затраченного времени

```
1  def test_simple_mult(A, B):
2  # Start the stopwatch / counter
3  t1_start = process_time()
4  for i in range(N_TEST):
5  simple_mult(A, M, B, N, M)
6  # Stop the stopwatch / counter
7  t1_stop = process_time()
```

3.4 Реализация алгоритмов

Листинг 3.2 демонстрирует реализацию классического алгоритма умножения.

Листинг 3.2 – Классический алгоритм умножения

```
1 def simple_mult(mat1, m, mat2, n, q):
2   res = np.zeros([m, q])
3   for i in range(m):
4     for j in range(q):
5       for k in range(n):
6         res[i][j] = res[i][j] + mat1[i][k] * mat2[k][j]
7   return res
```

Листинг 3.3 демонстрирует умножение матриц реализации алгоритмом Винограда.

Листинг 3.3 – Алгоритм умнложения Виноградом

```
def precompile\_rows\_win(mat, n, m):
  mh = np.zeros([n])
  for i in range(n):
    for j in range(m // 2):
      mh[i] = mh[i] + mat[i][j * 2] * mat[i][j * 2 + 1]
  return mh

def precompile\_cols\_win(mat, n, m):
  mv = np.zeros([m])

  for i in range(m):
    for j in range(n // 2):
      mv[i] = mv[i] + mat[j * 2][i] * mat[j * 2 + 1][i]
  return mv

def winograd\_mult(A, m, B, n, q):
  res = np.zeros([m, q])
  mh = precompile\_rows\_win(A, m, n)
  mv = precompile\_cols\_win(B, n, q)
  for i in range(m):
    for j in range(q):
    res[i][j] = -mh[i] - mv[j]
    for k in range(n // 2):
      res[i][j] = res[i][j] + (A[i][k*2] +
        B[k*2+1][j])*(A[i][k*2+1] + B[k*2][j])
  if n \% 2 != 0:
    for i in range(n):
      for j in range(m):
        res[i][j] = res[i][j] + A[i][n-1]*B[n-1][j]
  return res
```

Листинг 3.4 демонстрирует умножение реализации оптимизированным алгоритмом Винограда.

Листинг 3.4 – Оптимизированный алгоритм умножения Виноградом

```
def precompile_rows_win_opt(mat, n, m):
  mh = np.zeros([n])
  opt = m // 2
  for i in range(n):
    for j in range(opt):
      t = j << 1
      mh[i] += mat[i][t] * mat[i][t + 1]
  return mh

def precompile_cols_win(mat, n, m):
  mv = np.zeros([m])

  opt = n // 2
  for i in range(m):
    for j in range(opt):
      t = j << 1
      mv[i] += mat[t][i] * mat[t + 1][i]
  return mv

def winograd_mult_opt(A, m, B, n, q):
  res = np.zeros([m, q])
  mh = precompile_rows_win(A, m, n)
  mv = precompile_cols_win(B, n, q)

  opt = n // 2
  for i in range(m):
    for j in range(q):
      res[i][j] = -mh[i] - mv[j]
      for k in range(n // 2):
        t = k << 1
        res[i][j] += (A[i][t] + B[t+1][j])*(A[i][t+1] + B[t][j])
  if n % 2 != 0:
    for i in range(n):
      for j in range(m):
        res[i][j] += A[i][n-1]*B[n-1][j]
  return res
```

3.5 Тестовые данные

В таблице 3.1 представлены тестовые данные. Применена методология черного ящика. Все тесты пройдены успешно.

Таблица 3.1 – Функциональные тесты

1-ая матрица	2-ая матрица	Ожидаемый результат
$()$	$()$	Сообщение об ошибке
$()$	$\begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$	Сообщение об ошибке
$\begin{pmatrix} 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 0 & 1 \end{pmatrix}$	Сообщение об ошибке
$\begin{pmatrix} 5 \end{pmatrix}$	$\begin{pmatrix} 5 \end{pmatrix}$	$\begin{pmatrix} 25 \end{pmatrix}$
$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$	$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$	$\begin{pmatrix} 30 & 36 & 42 \\ 66 & 81 & 96 \\ 102 & 126 & 150 \end{pmatrix}$
$\begin{pmatrix} 4 & 3 \\ 2 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$	$\begin{pmatrix} 13 & 20 \\ 5 & 8 \end{pmatrix}$
$\begin{pmatrix} 1 & 2 & 3 \end{pmatrix}$	$\begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$	$\begin{pmatrix} 14 \end{pmatrix}$
$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$	$\begin{pmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{pmatrix}$	$\begin{pmatrix} 14 & 32 \\ 32 & 77 \\ 50 & 122 \end{pmatrix}$

Вывод

Было написано и протестировано программное обеспечение для решения поставленной задачи.

4 Исследовательская часть

В исследовательской части будут представлены время работы алгоритмов умножения матриц.

4.1 Технические характеристики

Замеры времени выполнялись на устройстве со следующими техническими характеристиками:

- операционная система Pop!_OS 22.04 LTS [5] Linux [6];
- оперативная память 16 Гб;
- процессор AMD® Ryzen 7 2700 eight-core processor × 16 [7].

Во время выполнения замеров времени устройство было подключено к блоку питания и не нагружено никакими приложениями, кроме встроенных приложений окружения, окружением и системой тестирования.

4.2 Демонстрация работы программы

На рисунке 4.1 представлен результат работы программы. Во время запуска генерируется матрица в размерах заданных внутри кода. В данном примере матрица размером 3 на 3 и выводится три матрицы, которые являются результатом умножением матриц алгоритмами: классическим, Винограда, оптимизированным алгоритмом Винограда.

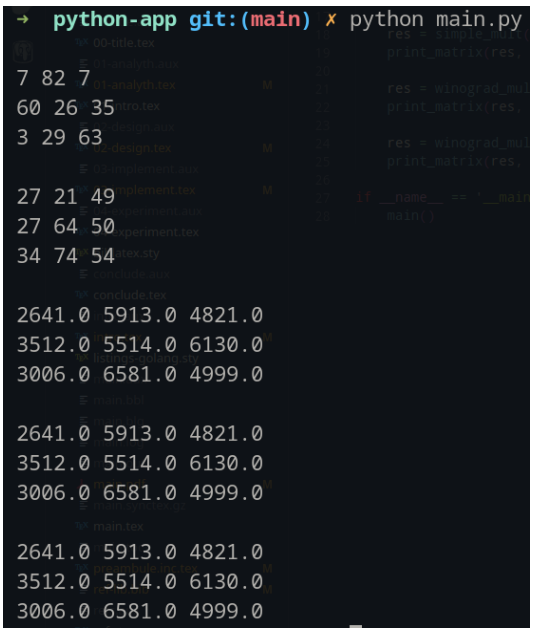

Рисунок 4.1 – Пример работы программы

4.3 Процессорное время выполнения реализации алгоритмов

Результаты профилирования реализации алгоритмов приведены в таблицах:

Таблица 4.1.1 – Время выполнения реализации алгоритмов при четной размерности матриц

n	Время, ns		
	Класс.	Виноград	Вин. опт.
10	4.49e-06	3.91e-06	4.01e-06
20	2.74e-05	2.24e-05	2.30e-05
30	8.84e-05	6.72e-05	7.01e-05
40	2.09e-04	1.49e-04	1.58e-04
50	3.96e-04	2.98e-04	3.14e-04
60	6.58e-04	4.95e-04	5.15e-04
70	1.07e-03	7.95e-04	8.43e-04
80	1.60e-03	1.20e-03	1.22e-03
90	2.26e-03	1.68e-03	1.77e-03
100	3.14e-03	2.25e-03	2.38e-03
150	1.06e-02	7.88e-03	8.33e-03
200	2.84e-02	2.20e-02	2.34e-02
250	6.73e-02	5.34e-02	5.74e-02
300	1.14e-01	8.97e-02	9.60e-02
350	1.80e-01	1.39e-01	1.50e-01
400	2.67e-01	2.05e-01	2.21e-01
450	4.17e-01	3.23e-01	3.48e-01
500	5.71e-01	4.42e-01	4.75e-01

Таблица 4.1.2 – Время выполнения реализации алгоритмов при нечетном размерности матриц

n	Время, ns		
	Класс.	Виноград	Вин. опт.
11	5.17e-06	4.56e-06	5.05e-06
21	3.25e-05	2.58e-05	2.59e-05
31	9.63e-05	7.23e-05	7.69e-05
41	2.14e-04	1.64e-04	1.73e-04
51	4.10e-04	3.15e-04	3.38e-04
61	7.13e-04	5.37e-04	5.70e-04
71	1.11e-03	8.36e-04	8.79e-04
81	1.67e-03	1.28e-03	1.35e-03
91	2.33e-03	1.73e-03	1.85e-03
101	3.23e-03	2.36e-03	2.49e-03
151	1.08e-02	8.03e-03	8.50e-03
201	2.74e-02	2.13e-02	2.29e-02
251	6.80e-02	5.27e-02	5.66e-02
301	1.16e-01	9.13e-02	9.82e-02
351	1.82e-01	1.40e-01	1.52e-01
401	2.58e-01	1.99e-01	2.14e-01
451	4.21e-01	3.26e-01	3.50e-01
501	5.74e-01	4.43e-01	4.76e-01

4.4 Результаты замеров времени выполнения реализаций алгоритмов

На рисунке 4.2 представлено время выполнения программы умножения квадратных матриц с четной линейной размерностью. На рисунке 4.3 представлено время выполнения программы умножения квадратных матриц с нечетной линейной размерностью.

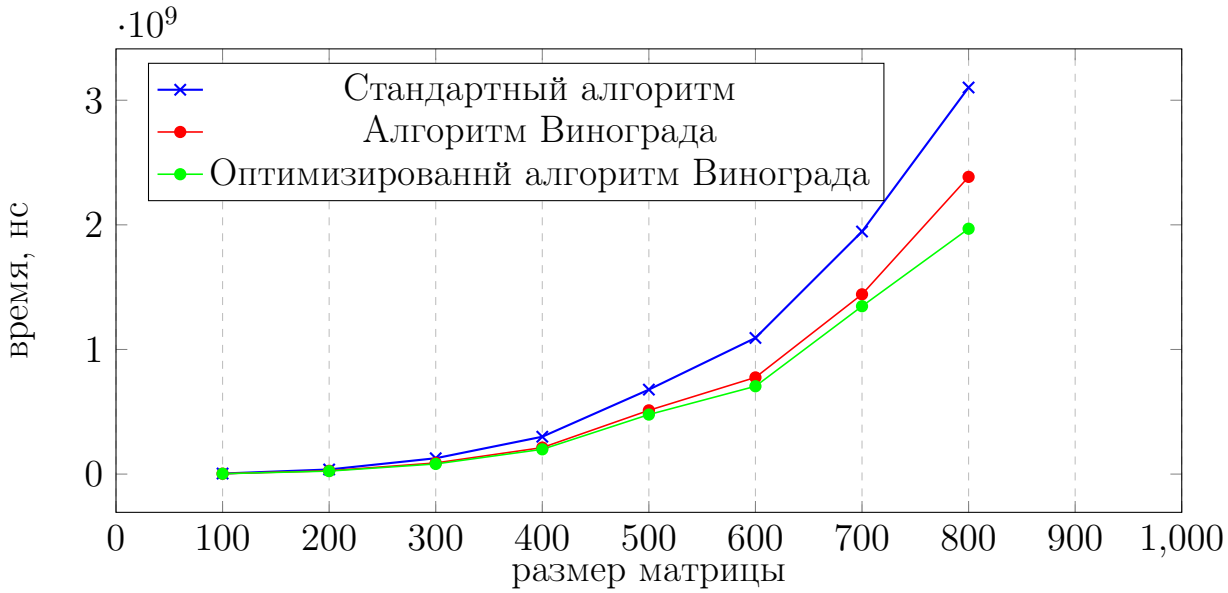

Рисунок 4.2 – Время выполнения (четная размерность матрицы)

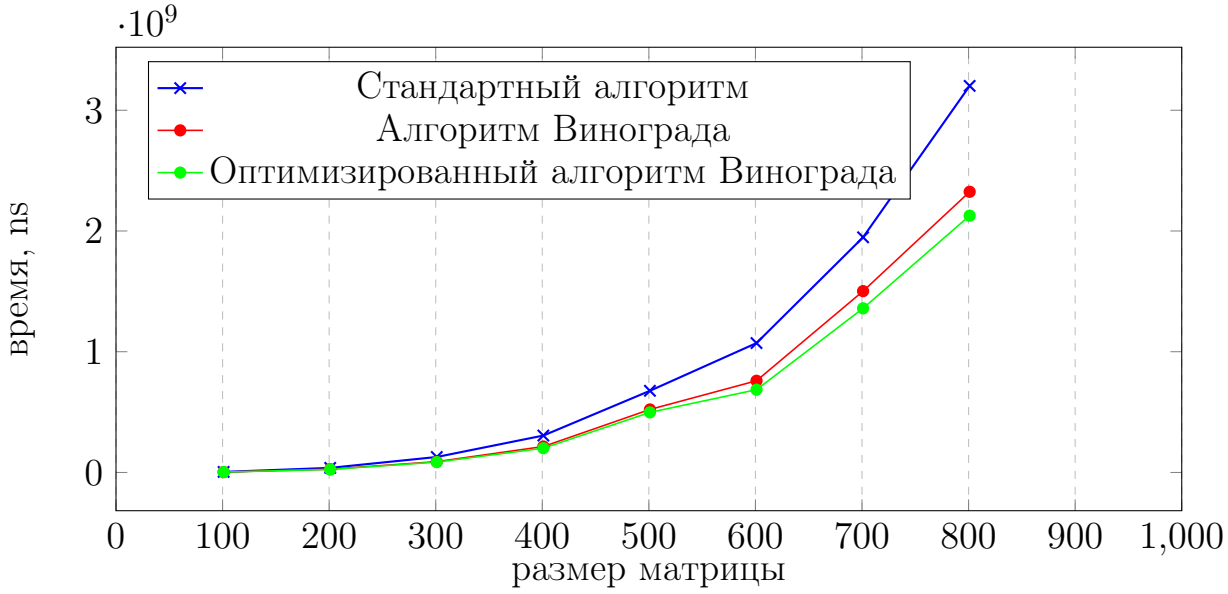

Рисунок 4.3 – Время выполнения (нечетная размерность матрицы)

Вывод

В данном разделе были сравнены реализации алгоритмов по затрачиваемому времени. Оптимизированный алгоритм Винограда является самым быстрым, за счет проведенных изменений в стандартном алгоритме Винограда.

Заключение

В ходе выполнения лабораторной работы были решены следующие задачи:

- были изучены и реализованы 3 алгоритма перемножения матриц: обычный, Винограда, модифицированный Винограда;
- был произведен анализ трудоёмкости алгоритмов на основе теоретических расчетов и выбранной модели вычислений;
- был сделан сравнительный анализ алгоритмов на основе экспериментальных данных;
- подготовлен отчет о лабораторной работе.

Оптимизированный алгоритм Винограда быстрее обычного на 5 процентов (на 0.1 нс) при размерах двух матриц 500 на 500.

Поставленная цель достигнута: алгоритмы умножения матриц изучены и проанализированы.

Список использованных источников

[1] Анисимов Н.С., Строганов Ю.В. Реализация алгоритма умножения матриц по Винограду на языке Haskell // Новые информационные технологии в автоматизированных системах: материалы двадцать первого научно-практического семинара. — М.: ИПМ им. М.В. Келдыша, 2018. — С. 390–395.

[2] Python Документация[Электронный ресурс]. Режим доступа: `https://docs.python.org/3/` (дата обращения: 24.09.2022).

[3] Vscode Документация[Электронный ресурс]. Режим доступа: `https://code.visualstudio.com/docs` (дата обращения: 24.09.2022).

[4] Функция process_time модуля time python [Электронный ресурс]. Режим доступа: `https://docs-python.ru/standart-library/modul-time-python/funktsija-process-time-modulja-time/` (дата обращения: 04.09.2022).

[5] Pop OS 22.04 LTS [Электронный ресурс]. Режим доступа: `https://pop.system76.com` (дата обращения: 04.09.2022).

[6] Linux – Документация [Электронный ресурс]. Режим доступа: `https://docs.kernel.org` (дата обращения: 24.09.2022).

[7] Процессор AMD® Ryzen 7 2700 eight-core processor × 16 [Электронный ресурс]. Режим доступа: `https://www.amd.com/en/products/cpu/amd-ryzen-7-2700` (дата обращения: 04.09.2022).