

*Государственное образовательное учреждение высшего  
профессионального образования*

**«Московский государственный технический  
университет имени Н.Э. Баумана»  
(МГТУ им. Н.Э. Баумана)**

---

ЛАБОРАТОРНАЯ РАБОТА №2  
ПО КУРСУ «АНАЛИЗ АЛГОРИТМОВ»

## **Умножение матриц**

Выполнил: Сорокин А.П., гр. ИУ7-52Б

Преподаватели: Волкова Л.Л., Строганов Ю.В.

*Москва, 2019 г.*

# Оглавление

|   |           |
|---|-----------|
| <b>Введение</b>   | <b>2</b>  |
| <b>1 Аналитическая часть</b>  | <b>3</b>  |
| 1.1 Задачи . . . . .  | 3         |
| 1.2 Описание алгоритмов . . . . .                                       | 3         |
| 1.2.1 Классический алгоритм умножения . . . . .                         | 3         |
| 1.2.2 Алгоритм Винограда . . . . .                                      | 4         |
| 1.2.3 Оптимизированный алгоритм Винограда . . . . .                     | 4         |
| 1.2.4 Модель вычислений . . . . .                                       | 5         |
| <b>2 Конструкторская часть</b>  | <b>6</b>  |
| 2.1 Схемы алгоритмов . . . . .  | 6         |
| 2.2 Оценка трудоёмкости . . . . .                                       | 9         |
| 2.2.1 Классический алгоритм . . . . .                                   | 9         |
| 2.2.2 Алгоритм Винограда . . . . .                                      | 9         |
| 2.2.3 Оптимизированный алгоритм Винограда . . . . .                     | 9         |
| 2.3 Список оптимизаций алгоритма Винограда . . . . .                    | 10        |
| 2.4 Замер используемой памяти . . . . .                                 | 10        |
| <b>3 Технологическая часть</b>  | <b>11</b> |
| 3.1 Требования к программному обеспечению . . . . .                     | 11        |
| 3.2 Средства реализации . . . . .                                       | 11        |
| 3.3 Реализации алгоритмов . . . . .                                     | 11        |
| 3.4 Тесты . . . . .   | 14        |
| <b>4 Экспериментальная часть</b>  | <b>15</b> |
| 4.1 Примеры работы . . . . .  | 15        |
| 4.2 Сравнение работы алгоритмов при чётных размерах матрицы . . . . .   | 15        |
| 4.3 Сравнение работы алгоритмов при нечётных размерах матрицы . . . . . | 16        |
| <b>Заключение</b>   | <b>18</b> |
| <b>Литература</b>   | <b>19</b> |

# Введение

В огромном количестве областей научной и технической сферы деятельности человека при различных математических расчетах используют такую операцию как умножение матриц. Это довольно трудоемкий процесс даже при небольших размерах матриц, так как требуется большое количество операций умножения и сложения различных чисел. По этой причине человек озадачен проблемой оптимизации умножения матриц и ускорения процесса вычисления.

Таким образом, эффективное умножение матриц по времени и затратам ресурсов является актуальной проблемой для науки и техники.

# 1. Аналитическая часть

## 1.1 Задачи

Цель лабораторной работы - изучение трех алгоритмов умножения матриц: классического, алгоритма Винограда и его оптимизации.

Для того чтобы добиться этой цели, были поставлены следующие задачи:

- изучить и реализовать классический алгоритм умножения матриц и алгоритм Винограда;
- оптимизировать работу алгоритма Винограда;
- выполнить сравнительный анализ трудоёмкостей алгоритмов;
- сравнить эффективность алгоритмов по времени и памяти.

## 1.2 Описание алгоритмов

### 1.2.1 Классический алгоритм умножения

Матрицей называют математический объект, эквивалентный двумерному массиву. Матрица является таблицей, на пересечении строк и столбцов находятся элементы матрицы. Количество строк и столбцов является размерностью матрицы.

Пусть даны две прямоугольные матрицы  $A$  и  $B$  размерности  $m \times n$ ,  $n \times q$  соответственно:

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix}$$

$$B = \begin{bmatrix} b_{11} & b_{12} & \dots & b_{1q} \\ b_{21} & b_{22} & \dots & b_{2q} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \dots & b_{nq} \end{bmatrix}$$

Тогда произведением матриц  $A$  и  $B$  называется матрица  $C$  размерностью  $m \times q$

$$C = \begin{bmatrix} c_{11} & c_{12} & \dots & c_{1q} \\ c_{21} & c_{22} & \dots & c_{2q} \\ \vdots & \vdots & \ddots & \vdots \\ c_{m1} & c_{m2} & \dots & c_{mq} \end{bmatrix}, [1] \quad (1.1)$$

в которой:

$$c_{ij} = \sum_{k=1}^n a_{ik} b_{kj} \quad (i = 1 \dots m; j = 1 \dots q).$$

### 1.2.2 Алгоритм Винограда

Исходя из равенства 1.1, видно, 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. [2] Рассмотрим два вектора  $U$  и  $V$ :

$$U = A_i = (u_1, u_2, \dots, u_n), \quad (1.2)$$

где  $U = A_i$  –  $i$ -ая строка матрицы  $A$ ,  
 $u_k = a_{ik}, k = 1 \dots n$  – элемент  $i$ -ой строки  $k$ -ого столбца матрицы  $A$ .

$$V = B_j = (v_1, v_2, \dots, v_n), \quad (1.3)$$

где  $V = B_j$  –  $j$ -ый столбец матрицы  $B$ ,  
 $v_k = b_{kj}, k = 1 \dots n$  – элемент  $k$ -ой строки  $j$ -ого столбца матрицы  $B$ .

По определению их скалярное произведение равно:

$$U \cdot V = u_1 v_1 + u_2 v_2 + u_3 v_3 + u_4 v_4. \quad (1.4)$$

Равенство 1.4 можно переписать в виде:

$$U \cdot V = (u_1 + v_2)(u_2 + v_1) + (u_3 + v_4)(u_4 + v_3) - u_1 u_2 - u_3 u_4 - v_1 v_2 - v_3 v_4. \quad (1.5)$$

В равенстве 1.4 насчитывается 4 операции умножения и 3 операции сложения, в равенстве 1.5 насчитывается 6 операций умножения и 9 операций сложения. Однако выражение  $-u_1 u_2 - u_3 u_4$  используются повторно при умножении  $i$ -ой строки матрицы  $A$  на каждый из столбцов матрицы  $B$ , а выражение  $-v_1 v_2 - v_3 v_4$  – при умножении  $j$ -ого столбца матрицы  $B$  на строки матрицы  $A$ . Таким образом, данные выражения можно вычислить предварительно для каждой строк и столбцов матриц для сокращения повторных вычислений. В результате повторно будут выполняться лишь 2 операции умножения и 7 операций сложения (2 операции нужны для добавления предварительно посчитанных произведений).

### 1.2.3 Оптимизированный алгоритм Винограда

Для оптимизации алгоритма Винограда могут использоваться такие стратегии, как:

- предварительные вычисления повторяющихся одинаковых действий;
- использование более быстрых операций при вычислении (такие, как сдвиг битов вместо умножения или деления на 2);
- уменьшения количества повторных проверок;
- использование аналогичных конструкций, уменьшающих трудоёмкость операций (к примеру, замена сложения с 1 на инкремент).

Ниже представлен список личностей, проводивших оптимизацию алгоритма:

- в 2010 Эндрю Стотерс усовершенствовал алгоритм до  $O(n^{2.374})$ ;
- в 2011 году Вирджиния Уильямс усовершенствовала алгоритм до  $O(n^{2.3728642})$ ;
- в 2014 году Франсуа Ле Галль упростил метод Уильямса и получил новую улучшенную оценку  $O(n^{2.3728639})$ .

#### 1.2.4 Модель вычислений

В рамках данной работы используется следующая модель вычислений:

- операции, имеющие трудоемкость 1:  $<$ ,  $>$ ,  $=$ ,  $<=$ ,  $=>$ ,  $==$ ,  $!=$ ,  $+$ ,  $-$ ,  $*$ ,  $/$ ,  $+=$ ,  $-=$ ,  $*=$ ,  $/=$ ,  $[]$ ;
- оператор условного перехода имеет трудоёмкость, равную трудоёмкости операторов тела условия;
- оператор цикла `for` имеет трудоемкость:

$$F_{for} = F_{init} + F_{check} + N * (F_{body} + F_{inc} + F_{check}), \quad (1.6)$$

где  $F_{init}$  – трудоёмкость инициализации,  $F_{check}$  – трудоёмкость проверки условия,  $F_{inc}$  – трудоёмкость инкремента аргумента,  $F_{body}$  – трудоёмкость операций в теле цикла,  $N$  – число повторений. [3]

## 2. Конструкторская часть

### 2.1 Схемы алгоритмов

На рисунках 2.1, 2.2, 2.3 представлены схемы алгоритмов трёх алгоритмов умножения матриц.

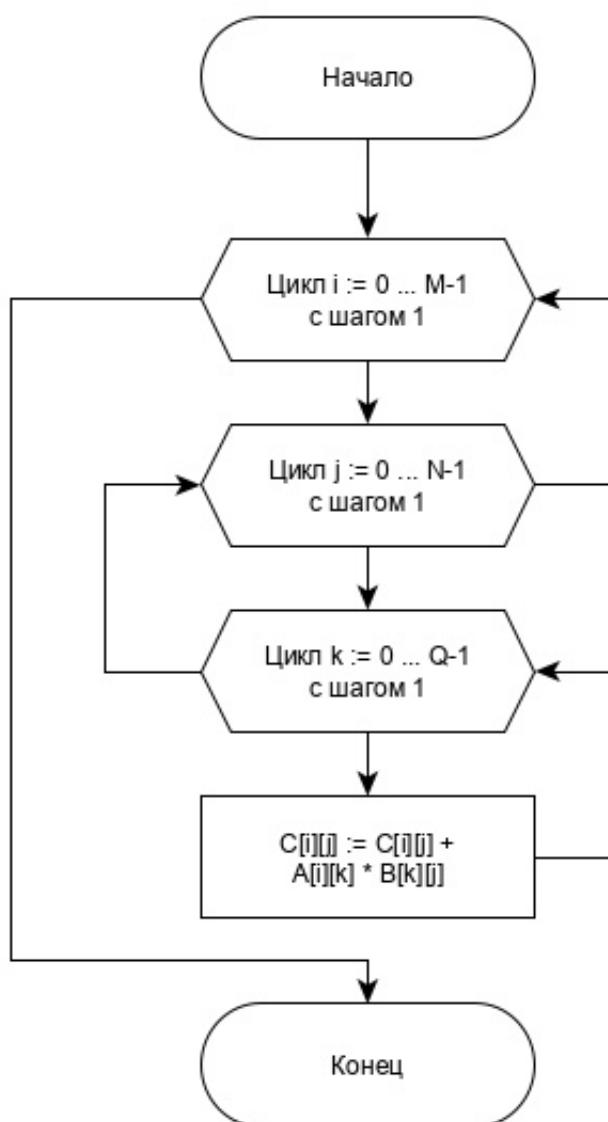


Рис. 2.1: Классический алгоритм

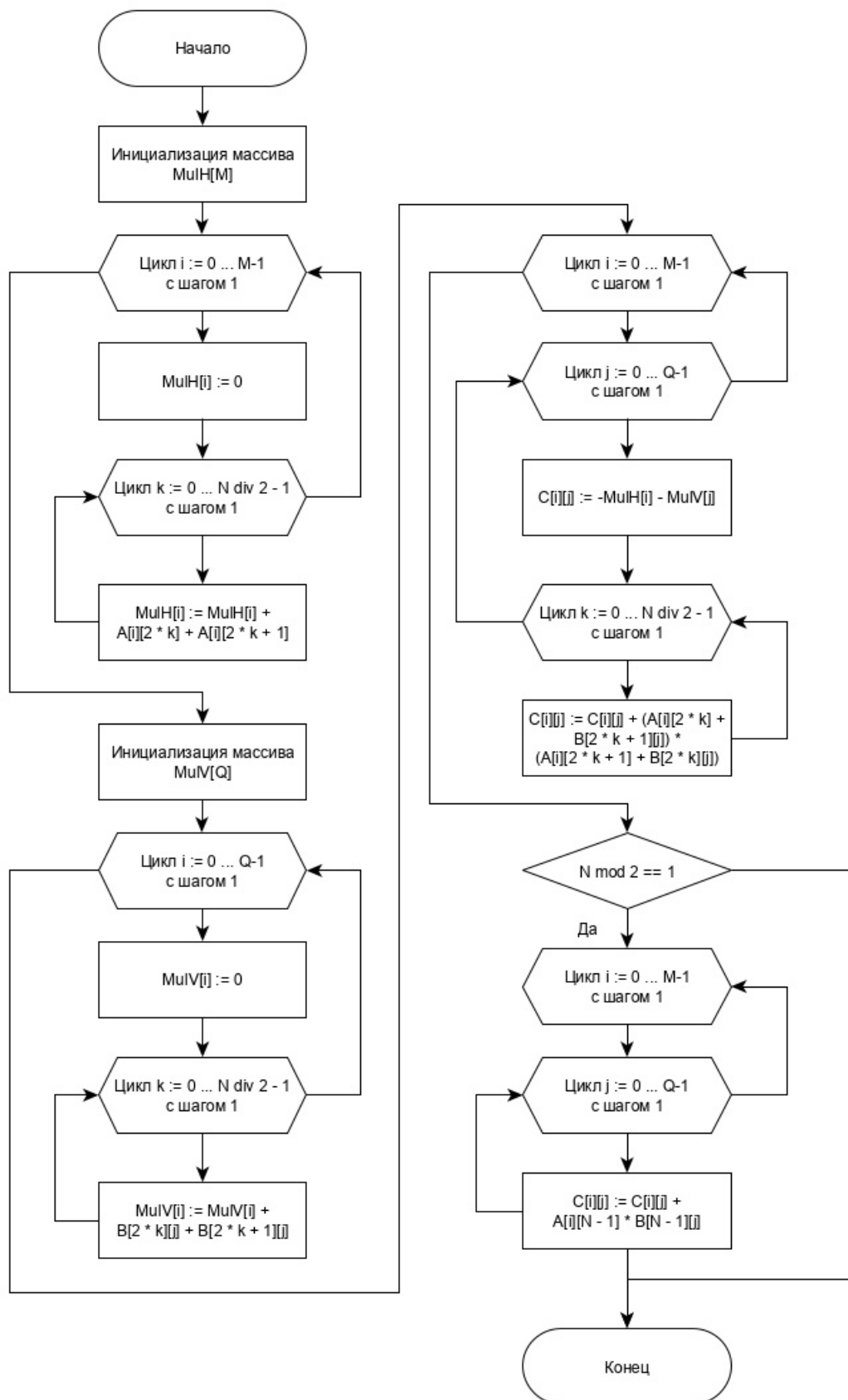


Рис. 2.2: Алгоритм Винограда



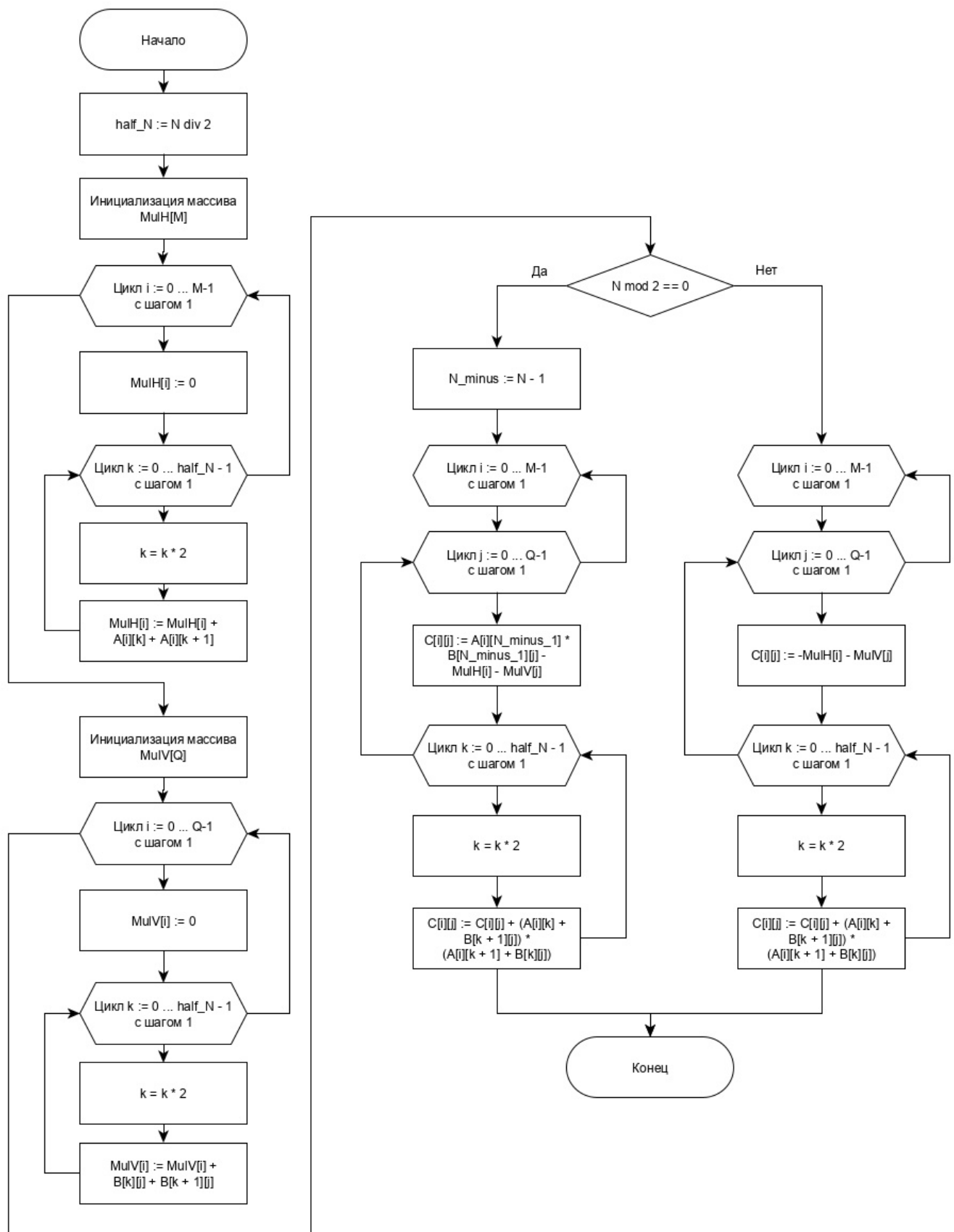


Рис. 2.3: Оптимизированный алгоритм Винограда

## 2.2 Оценка трудоёмкости

Пусть даны две матрицы  $A$  и  $B$  размерностью  $M \times N$  и размерностью  $N \times Q$  соответственно. Рассмотрим трудоёмкость трёх алгоритмов умножения матриц.

### 2.2.1 Классический алгоритм

$$f_{classic} = 13MNQ + 4MQ + 4M + 1 \quad (2.1)$$

### 2.2.2 Алгоритм Винограда

Трудоёмкости составных частей алгоритма:

- цикл создания вектора MulH:  $f_I = \frac{15}{2}MN + 4M + 2$ ;
- цикл создания вектора MulV:  $f_{II} = \frac{15}{2}NQ + 4Q + 2$ ;
- основной цикл:  $f_{III} = 13MNQ + 4MQ + 4M + 2$ ;
- условный переход при чётном  $N$ :  $f_{IV} = 2$ ;
- условный переход и цикл при нечётном  $N$ :  $f_{IV} = 17MQ + 4M + 4$ .

Общая трудоёмкость алгоритма:

- при чётном  $N$ :

$$f_{vin} = f_I + f_{II} + f_{III} + f_{IV} = 13MNQ + \frac{15}{2}MN + \frac{15}{2}NQ + 4MQ + 8M + 4Q + 8 \quad (2.2)$$

- при нечётном  $N$ :

$$f_{vin} = f_I + f_{II} + f_{III} + f_{IV} = 13MNQ + \frac{15}{2}MN + \frac{15}{2}NQ + 21MQ + 12M + 4Q + 10 \quad (2.3)$$

### 2.2.3 Оптимизированный алгоритм Винограда

Трудоёмкости составных частей алгоритма:

- цикл создания вектора MulH:  $f_I = \frac{11}{2}MN + 4M + 2$ ;
- цикл создания вектора MulV:  $f_{II} = \frac{11}{2}NQ + 4Q + 2$ ;
- основной цикл при чётном  $N$ :  $f_{III} = 10MNQ + 17MQ + 4M + 6$ ;
- основной цикл при нечётном  $N$ :  $f_{III} = 10MNQ + 10NQ + 4M + 2$ .

Общая трудоёмкость алгоритма:

- при чётном  $N$ :

$$f_{opt} = f_I + f_{II} + f_{III} = 10MNQ + \frac{11}{2}MN + \frac{11}{2}NQ + 17MQ + 8M + 4Q + 10 \quad (2.4)$$

- при нечётном  $N$ :

$$f_{opt} = f_I + f_{II} + f_{III} = 10MNQ + \frac{11}{2}MN + \frac{11}{2}NQ + 10NQ + 8M + 4Q + 6 \quad (2.5)$$

## 2.3 Список оптимизаций алгоритма Винограда

Алгоритм Винограда был оптимизирован с помощью следующих модификаций:

1. Замена конструкций вида  $a = a + b$  (трудоемкость = 2) на  $a += b$  (трудоемкость = 1).
2. Вычисление повторно используемых величин выполняется предварительно ( $N/2, N - 1, 2 * k$ ).
3. Цикл вычислений для нечётных элементов включён в основной цикл, добавив дополнительные операции. Данная модификация исключает повторные проверки на нечётность  $N$ .

## 2.4 Замер используемой памяти

Пусть даны две матрицы  $A$  и  $B$  размерностью  $M \times N$  и размерностью  $N \times Q$  соответственно и для хранения целого числа требуется 4 байта памяти.

В каждом из алгоритмов требуется хранить исходные матрицы  $A$  и  $B$  и матрицу результата умножения  $C$ , которая имеет размеры  $M \times Q$ . Таким образом, под хранение матриц требуется  $4 \cdot (MN + NQ + MQ)$  байт памяти.

Для алгоритма Винограда и его оптимизированного варианта требуется также хранить два дополнительных вектора  $MulH$  и  $MulV$ . Их размеры  $M$  и  $Q$  соответственно, следовательно требуется дополнительно  $4 \cdot (M + Q)$  байт памяти. В итоге для алгоритмов Винограда требуется  $4 \cdot (MN + NQ + MQ + M + Q)$  байт памяти. Таким образом, при больших размерах матриц (больше 100) алгоритмы Винограда будут значительно проигрывать по памяти классическому алгоритму.

## 3. Технологическая часть

### 3.1 Требования к программному обеспечению

На вход подаются размеры двух матриц. Матрицы генерируются случайным образом и выводятся на экран. На выход программа выдаёт три матрицы, которые являются результатами работы трёх различных алгоритмов умножения.

### 3.2 Средства реализации

Для реализации программы был использован язык C++ [4]. Для замера процессорного времени была использована функция `rdtsc()` из библиотеки `stdrin.h`.

### 3.3 Реализации алгоритмов

В листингах 3.1, 3.2, 3.3 представлены коды реализации алгоритмов умножения матриц.

Листинг 3.1: Классический алгоритм

```
1 void multiply_classic(int **A, int **B, int **C, unsigned M, unsigned N, unsigned Q)
2 {
3     for (unsigned i = 0; i < M; i++)
4         for (unsigned j = 0; j < Q; j++)
5             {
6                 C[i][j] = 0;
7                 for (unsigned k = 0; k < N; k++)
8                     C[i][j] += A[i][k] * B[k][j];
9             }
10 }
```

Листинг 3.2: Алгоритм Винограда

```
1 void multiply_vinograd(int **A, int **B, int **C, unsigned M, unsigned N, unsigned Q)
2 {
3     int *MulH = new int[M];
4     for (unsigned i = 0; i < M; i++)
5         {
6             MulH[i] = 0;
7             for (unsigned k = 0; k < N / 2; k++)
8                 MulH[i] = MulH[i] + A[i][2 * k] * A[i][2 * k + 1];
9         }
10
11     int *MulV = new int[Q];
12     for (unsigned i = 0; i < Q; i++)
13         {
```

```

14     MulV[i] = 0;
15     for (unsigned k = 0; k < N / 2; k++)
16         MulV[i] = MulV[i] + B[2 * k][i] * B[2 * k + 1][i];
17 }
18
19 for (unsigned i = 0; i < M; i++)
20     for (unsigned j = 0; j < Q; j++)
21     {
22         C[i][j] = -MulH[i] - MulV[j];
23         for (unsigned k = 0; k < N / 2; k++)
24             C[i][j] = C[i][j] + (A[i][2 * k] + B[2 * k + 1][j]) *
25                 (A[i][2 * k + 1] + B[2 * k][j]);
26     }
27
28
29 if (N % 2 == 1)
30     for (unsigned i = 0; i < M; i++)
31         for (unsigned j = 0; j < Q; j++)
32             C[i][j] = C[i][j] + A[i][N - 1] * B[N - 1][j];
33
34 delete [] MulH;
35 delete [] MulV;
36 }

```

Листинг 3.3: Оптимизированный алгоритм Винограда

```

1 void multiply_vinograd_opt(int **A, int **B, int **C, unsigned M, unsigned N, unsigned Q)
2 {
3     int *MulH = new int[M];
4     for (unsigned i = 0; i < M; i++)
5     {
6         MulH[i] = 0;
7         for (unsigned k = 0; k < N; k <= 1)
8             MulH[i] += A[i][k] * A[i][k + 1];
9     }
10
11     int *MulV = new int[Q];
12     for (unsigned i = 0; i < Q; i++)
13     {
14         MulV[i] = 0;
15         for (unsigned k = 0; k < N; k <= 1)
16             MulV[i] += B[k][i] * B[k + 1][i];
17     }
18 }
19
20 if (N % 2)
21 {
22     unsigned N_minus_1 = N - 1;
23     for (unsigned i = 0; i < M; i++)
24         for (unsigned j = 0; j < Q; j++)
25         {
26             C[i][j] = A[i][N_minus_1] * B[N_minus_1][j] - MulH[i] - MulV[j];
27             for (unsigned k = 0; k < N; k <= 1)
28                 C[i][j] += (A[i][k] + B[k + 1][j]) * (A[i][k + 1] + B[k][j]);

```

```

29     }
30 }
31 else
32 {
33     for (unsigned i = 0; i < M; i++)
34         for (unsigned j = 0; j < Q; j++)
35         {
36             C[i][j] = -MulH[i] - MulV[j];
37             for (unsigned k = 0; k < N; k <= 1)
38                 C[i][j] += (A[i][k] + B[k + 1][j]) * (A[i][k + 1] + B[k][j]);
39         }
40 }
41
42 delete [] MulH;
43 delete [] MulV;
44 }

```

### 3.4 Тесты

Для проверки корректности работы были подготовлены функциональные тесты, представленные в таблице 3.1. Входные данные удовлетворяют условиям, необходимым для умножения матриц, так как проверка на соответствие их размеров возложена на другую функцию.

Таблица 3.1: Функциональные тесты

| Матрица 1   | Матрица 2  | Ожидание  |
|---|--|---|
| $[5]$   | $[-8]$   | $[-40]$   |
| $\begin{bmatrix} 2 & 1 & 1 \end{bmatrix}$                 | $\begin{bmatrix} 1 \\ -1 \\ 5 \end{bmatrix}$             | $[6]$   |
| $\begin{bmatrix} 5 & 1 \\ 0 & -1 \end{bmatrix}$           | $\begin{bmatrix} 3 & -5 \\ 10 & 0 \end{bmatrix}$         | $\begin{bmatrix} -10 & 25 \\ -10 & 0 \end{bmatrix}$ |
| $\begin{bmatrix} 1 & 2 & 0 \\ 3 & 0 & -1 \end{bmatrix}$   | $\begin{bmatrix} 1 & 2 \\ 3 & 0 \\ 0 & -2 \end{bmatrix}$ | $\begin{bmatrix} 7 & 2 \\ 3 & 8 \end{bmatrix}$      |
| $\begin{bmatrix} 1 & 1 & -1 \\ 5 & -3 & -4 \end{bmatrix}$ | $\begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}$  | $\begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$      |
| $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$            | $\begin{bmatrix} 1 & 3 \\ -2 & 1 \end{bmatrix}$          | $\begin{bmatrix} 1 & 3 \\ -2 & 1 \end{bmatrix}$     |

В результате проверки реализации всех алгоритмов умножения прошли все поставленные функциональные тесты.

## 4. Экспериментальная часть

### 4.1 Примеры работы

На рисунке 4.1 представлен пример работы программы, демонстрирующий корректную работу алгоритмов.

```
Enter 3 size of matrices (M, N, Q): 5 5 5

-8 6 -5 0 5
-7 6 5 -4 0
-1 2 -2 -6 -7
7 6 -7 0 2
2 5 10 6 -9

-5 -5 -7 5 6
8 -7 10 5 -3
5 4 -2 10 10
7 -6 -6 5 -7
2 -4 -6 -1 -1

73 -42 96 -65 -121
80 37 123 25 18
-45 47 109 -38 17
-18 -113 13 -7 -48
104 -5 34 174 64

73 -42 96 -65 -121
80 37 123 25 18
-45 47 109 -38 17
-18 -113 13 -7 -48
104 -5 34 174 64

73 -42 96 -65 -121
80 37 123 25 18
-45 47 109 -38 17
-18 -113 13 -7 -48
104 -5 34 174 64
```

Рис. 4.1: Пример работы программы

### 4.2 Сравнение работы алгоритмов при чётных размерах матрицы

Для сравнения времени работы алгоритмов умножения матриц были использованы квадратные матрицы размером от 100 до 1000 с шагом 100. Эксперимент для более точного результата повторялся 100 раз. Итоговый результат рассчитывался как средний из полученных результатов. Результаты измерений показаны в таблице 4.1 и на рисунке 4.2.



Таблица 4.1: Время работы алгоритмов при чётных размерах матриц в тактах процессора

| Размер матриц | Классический | Алг-м Винограда | Оптимиз. алг-м Винограда |
|---------------|--------------|-----------------|--------------------------|
| 100           | 9832603      | 7754245         | 7744877                  |
| 200           | 83641431     | 66133497        | 65139523                 |
| 300           | 275961612    | 222657307       | 212578053                |
| 400           | 687812610    | 554745864       | 524447105                |
| 500           | 1445622837   | 1145269689      | 1108285653               |
| 600           | 2535128351   | 2023845576      | 1970742658               |
| 700           | 4109751947   | 3253209027      | 3200000610               |
| 800           | 6189672669   | 4885014816      | 4708955900               |
| 900           | 9779595722   | 7702124130      | 7606387468               |
| 1000          | 21459177654  | 17580178871     | 16442692055              |

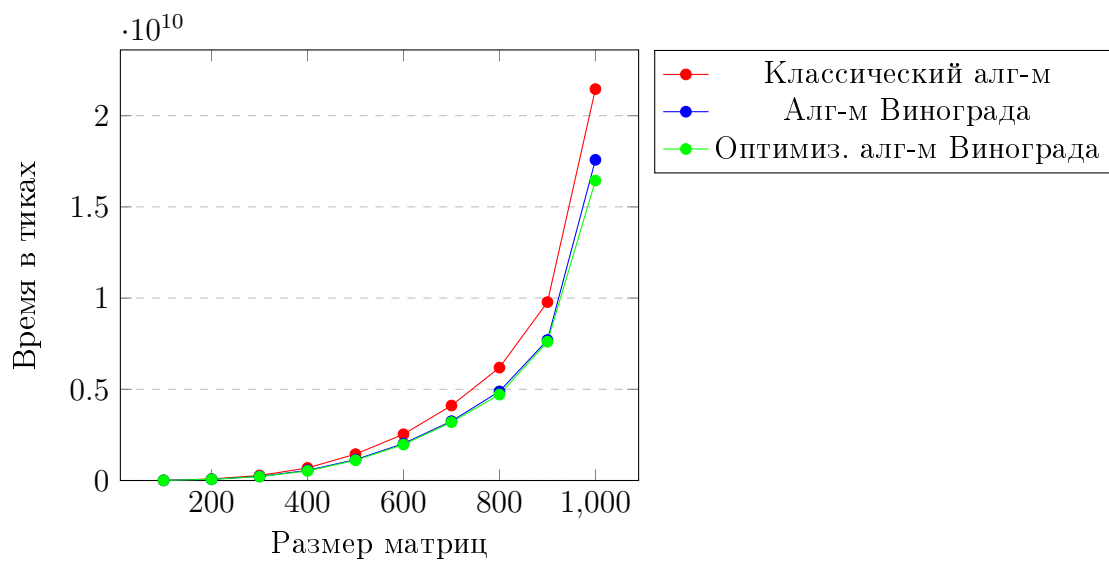


Рис. 4.2: График времени работы алгоритмов при чётных размерах матриц

Из результатов экспериментов можно сделать вывод о том, что алгоритм Винограда выигрывает классический алгоритм умножения матриц в среднем на 18%. Оптимизированный алгоритм работает незначительно, но быстрее обычного алгоритма Винограда.

### 4.3 Сравнение работы алгоритмов при нечётных размерах матрицы

Для сравнения времени работы алгоритмов умножения матриц были использованы квадратные матрицы размером от 101 до 1001 с шагом 100. Эксперимент для более точного результата повторялся 100 раз. Итоговый результат рассчитывался как средний из полученных результатов. Результаты измерений показаны в таблице 4.2 и на рисунке 4.3.

Таблица 4.2: Время работы алгоритмов при нечётных размерах матриц в тактах процессора

| Размер матриц | Классический | Алг-м Винограда | Оптимиз. алг-м Винограда |
|---------------|--------------|-----------------|--------------------------|
| 101           | 10547508     | 8230191         | 8177305                  |
| 201           | 82367567     | 65381968        | 64826867                 |
| 301           | 336340052    | 266920865       | 258607497                |
| 401           | 812787248    | 651249088       | 645290661                |
| 501           | 1835189141   | 1467674147      | 1409757023               |
| 601           | 3007503824   | 2395686951      | 2325860775               |
| 701           | 4931828183   | 3919747517      | 3852576801               |
| 801           | 9483733814   | 7620831752      | 7551663392               |
| 901           | 13990997406  | 11309714724     | 10896364853              |
| 1001          | 23043318998  | 18788207855     | 18045078361              |

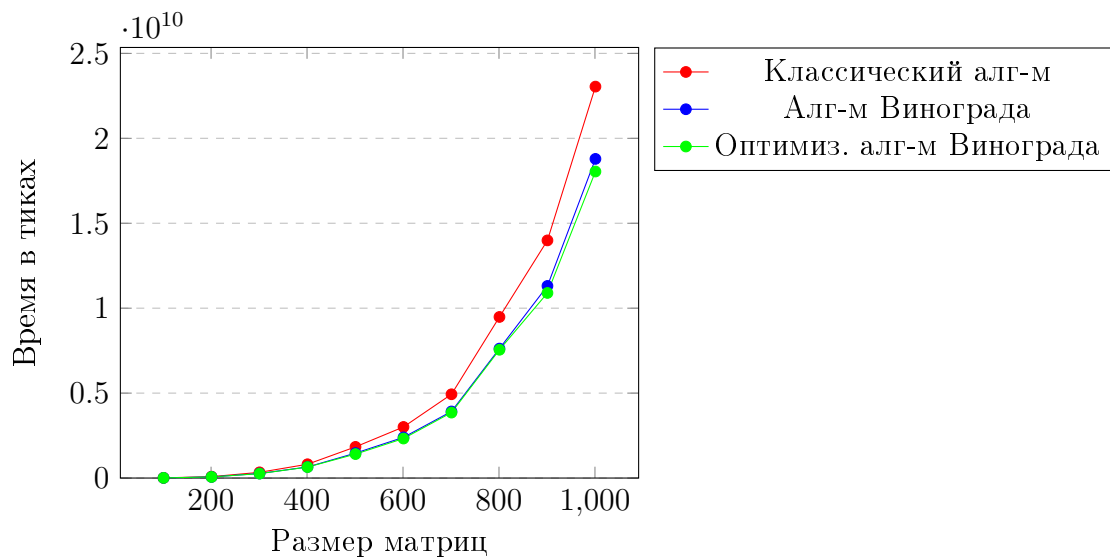


Рис. 4.3: График времени работы алгоритмов при нечётных размерах матриц

Для случая с нечётными размерами матриц можно сделать те же выводы, что и для случая с чётными. При этом можно заметить, что классический алгоритм в среднем работает за то же время, что и при чётных размерах, в то время как алгоритм Винограда и его оптимизация работают дольше за счёт дополнительных операций при нечётном случае. Однако по-прежнему классический алгоритм проигрывает по времени на те же величины.

## Заключение

В ходе лабораторной работе были изучены и реализованы три алгоритма умножения матриц: классический алгоритм, алгоритм Винограда и его оптимизированный вариант. Сравнительный анализ алгоритмов показал, что алгоритмы Винограда, введя дополнительные векторы и увеличив тем самым расход памяти, добились уменьшения времени выполнения умножения за счёт уменьшения трудоёмких операций: неоптимизированный и оптимизированный варианты работают на 18-20% быстрее классического алгоритма.

## Литература

- [1] Бахвалов, Н.С. Численные методы / Н.С. Бахвалов, Н.П. Жидков, Г.М. Кобельков – М.: Наука, 1987.
- [2] Jelfimova L. A new fast systolic array for modified Winograd algorithm // Proc. Sevens Int. Workshop on Parallel Processing by Cellular Automata and Array, PARCELLA-96 (Berlin, Germany, Sept. 1996). — Berlin: Akad. Verlag. — 1996.
- [3] Кормен, Т. Алгоритмы: построение и анализ / Т. Кормен, Ч. Лейзерсон, Р.М. Ривест: – МЦНТО, 1999.
- [4] <https://cppreference.com/> [Электронный ресурс]