Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное 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 № 1
по дисциплине «Анализ алгоритмов»

Тема Алгоритмы умножения матриц

Студент Доколин Г. А.

Группа ИУ7-52Б

Преподаватели Волкова Л. Л.

Москва, 2025

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В рамках данной лабораторной работы анализируются алгоритмы умножения матриц: стандартный и метод Винограда, который рассматривается в базовой и оптимизированной модификациях. Целью исследования является сравнительный анализ производительности и особенностей реализации указанных алгоритмов. Для достижения цели были поставлены следующие задачи:

1) изложить математические принципы, лежащие в основе классического метода умножения матриц и алгоритма Винограда.
2) представить схемы, описывающие логику работы трёх рассматриваемых алгоритмов: стандартного, базового Винограда и его оптимизированной версии.
3) сформулировать вычислительную модель и провести в её рамках теоретический анализ трудоёмкости трёх изучаемых алгоритмов умножения матриц.
4) разработать программные реализации трёх алгоритмов умножения.
5) провести исследование по измерению процессорного времени, затрачиваемого выполнением реализованных алгоритмов.
6) выполнить сравнительную характеристику алгоритмов.

1 Аналитическая часть

В данном разделе излагаются математические принципы, лежащие в основе двух методов умножения матриц: классического и алгоритма Винограда.

1.1 Матрица

Матрица — это прямоугольная таблица чисел, содержащая $n \times m$ элементов, расположенных в n строках и m столбцах [1].

Матрица обозначается следующим образом:

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix}, \tag{1.1}$$

где

a_{ij} — элементы матрицы ($i = \overline{1, m}; j = \overline{1, n}$);

i — номер строки;

j — номер столбца;

$m \times n$ — размер матрицы.

При работе с матрицами выделяют три основные операции:

1) сложение, данное действие выполняется для двух матриц одинакового размера;

2) умножение, данное действие выполняется для двух таких матриц, что у первой матрицы число столбцов должно быть равно числу строк второй матрицы;

3) транспонирование, данное действие выполняется для матриц любых размеров.

1.2 Стандартный алгоритм

Пусть даны две матрицы. Матрица A, размерностью $m \times n$ и матрица B, размерностью $n \times k$:

$$A_{m \times n} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix}, B_{n \times k} = \begin{pmatrix} b_{11} & b_{12} & \cdots & b_{1k} \\ b_{21} & b_{22} & \cdots & b_{2k} \\ \cdots & \cdots & \cdots & \cdots \\ b_{n1} & b_{n2} & \cdots & b_{nk} \end{pmatrix}, \tag{1.2}$$

тогда матрица C будет иметь размер $m \times k$:

$$C_{m \times k} = \begin{pmatrix} c_{11} & c_{12} & \cdots & c_{1k} \\ c_{21} & c_{22} & \cdots & c_{2k} \\ \cdots & \cdots & \cdots & \cdots \\ c_{m1} & c_{m2} & \cdots & c_{mk} \end{pmatrix}, \tag{1.3}$$

где

$$c_{ij} = \sum_{t=1}^{n} a_{it} \cdot b_{tj},\ i = \overline{1,m}; j = \overline{1,k} \tag{1.4}$$

1.3 Алгоритм Винограда

В формуле (1.4) каждый элемент результирующей матрицы равен скалярному произведению соответствующих строк и столбцов исходных матриц. Пусть даны два вектора $\overrightarrow{V} = (v_1, v_2, \ldots, v_n)$ и $\overrightarrow{U} = (u_1, u_2, \ldots, u_n)$, тогда их скалярное произведение будет записано в виде:

$$\overrightarrow{V} \cdot \overrightarrow{U} = v_1 \cdot u_1 + v_2 \cdot u_2 + \cdots + v_n \cdot u_n \tag{1.5}$$

Это равенство (1.5) представляется в виде выражения для чётных n (1.6) и в виде выражения для нечётных n (1.7) :

$$\begin{aligned} \overrightarrow{V} \cdot \overrightarrow{U} = (v_1 + u_2) \cdot (v_2 + u_1) + \cdots (v_{n-1} + u_n) \cdot (v_n + u_{n-1}) - \\ - v_1 \cdot v_2 - u_1 \cdot u_2 - \cdots - v_{n-1} \cdot v_n - u_{n-1} \cdot u_n \end{aligned} \tag{1.6}$$

$$\begin{aligned} \overrightarrow{V} \cdot \overrightarrow{U} = (v_1 + u_2) \cdot (v_2 + u_1) + \cdots (v_{n-2} + u_{n-1}) \cdot (v_{n-1} + u_{n-2}) - \\ - v_1 \cdot v_2 - u_1 \cdot u_2 - \cdots - v_{n-2} \cdot v_{n-1} - u_{n-2} \cdot u_{n-1} + v_n \cdot u_n \end{aligned} \tag{1.7}$$

Произведения вида $q_i \cdot q_{i+1}$ предварительно вычисляются для каждой матрицы: для первой матрицы рассчитываются произведения элементов по строкам, для второй — по столбцам.

1.4 Оптимизированный алгоритм Винограда

В алгоритме Винограда дополнительное сокращение числа арифметических операций достигается за счёт вынесения части вычислений за пределы вложенных циклов. Оптимизация основана на двух ключевых приёмах:

1) **предварительное вычисление постоянных выражений.** Значения, зависящие только от размеров матриц (например, $N - 1$, $\lfloor N/2 \rfloor$), вычисляются один раз перед началом умножения. Это избавляет от повторных вычислений в теле циклов.

2) **буферизация промежуточных результатов.** При заполнении массивов дополнительно формируются вспомогательные буферы:

— для каждой строки первой матрицы сохраняются суммы произведений пар элементов;

— для каждого столбца второй матрицы аналогично сохраняются накопленные значения.

Использование буферов позволяет уменьшить количество умножений и сложений при вычислении каждого элемента результирующей матрицы.

Таким образом, оптимизированный алгоритм Винограда, сохраняя идею скалярного произведения строк и столбцов, уменьшает общее количество выполняемых операций. Это делает

его более эффективным при работе с матрицами больших размеров.

Вывод

В аналитической части были рассмотрены математические основы алгоритма Винограда и стандартного алгоритма умножения матриц.

2 Конструкторская часть

В данной главе представлены требования к разрабатываемому программному обеспечению, приведены схемы алгоритмов и введена модель вычислений для проведения теоретической оценки трудоёмкости.

2.1 Требования к программе

Для разрабатываемой программы определены следующие задачи:

1) реализовать интерфейс выбора операций для пользователя;
2) обеспечить возможность работы программы в двух режимах: одиночное выполнение и массовое измерение времени выполнения;
3) в режиме одиночного запуска предусмотреть:
   - ввод размеров и элементов двух матриц;
   - проверку корректности введённых данных;
   - проверку совместимости матриц для операции умножения;
4) в режиме массового измерения времени фиксировать затраченное процессорное время и выводить результаты в табличном виде.

2.2 Разработка алгоритмов

На рисунке 2.1 представлена схема стандартного алгоритма перемножения матриц. Рисунки 2.2–2.3 демонстрируют структуру алгоритма Винограда.

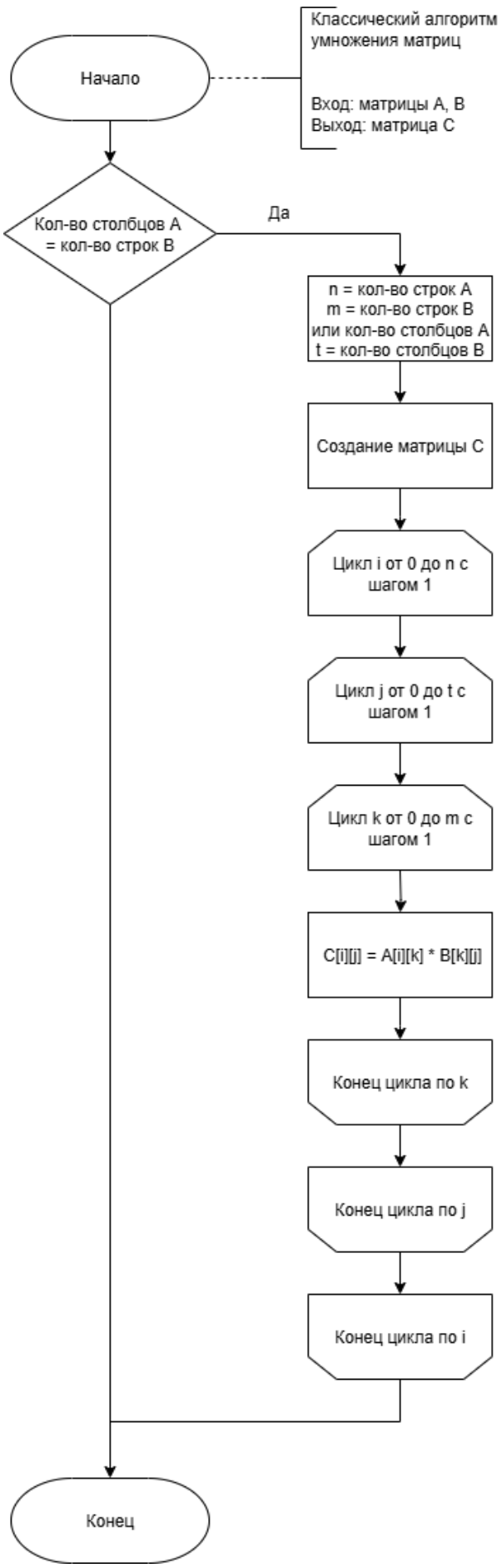

Рисунок 2.1 — Схема стандартного алгоритма умножения матриц

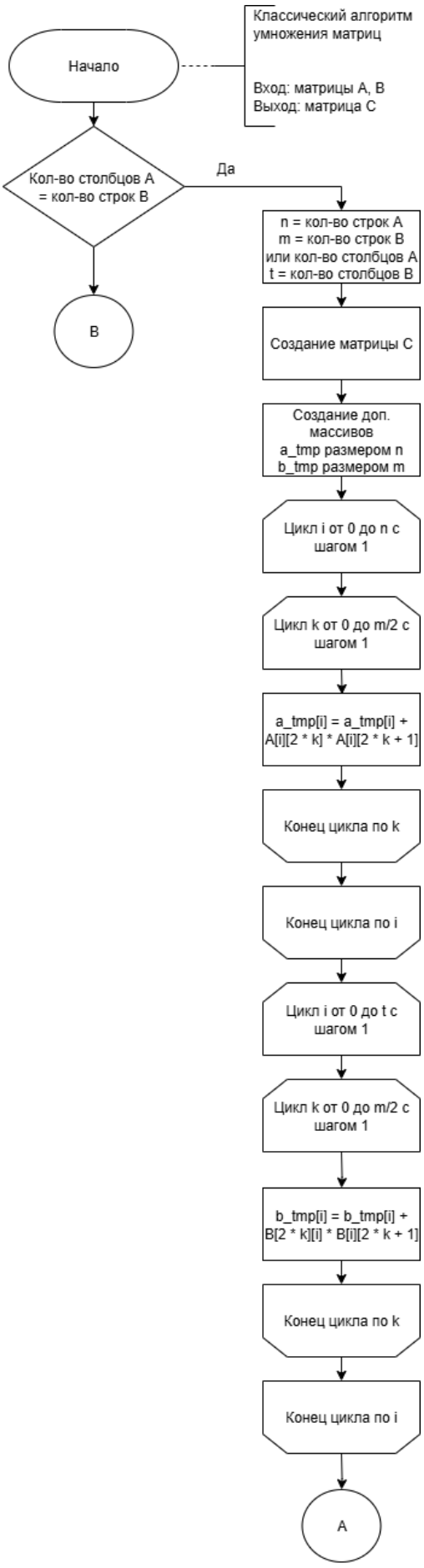

Рисунок 2.2 — Схема алгоритма Винограда (часть 1)

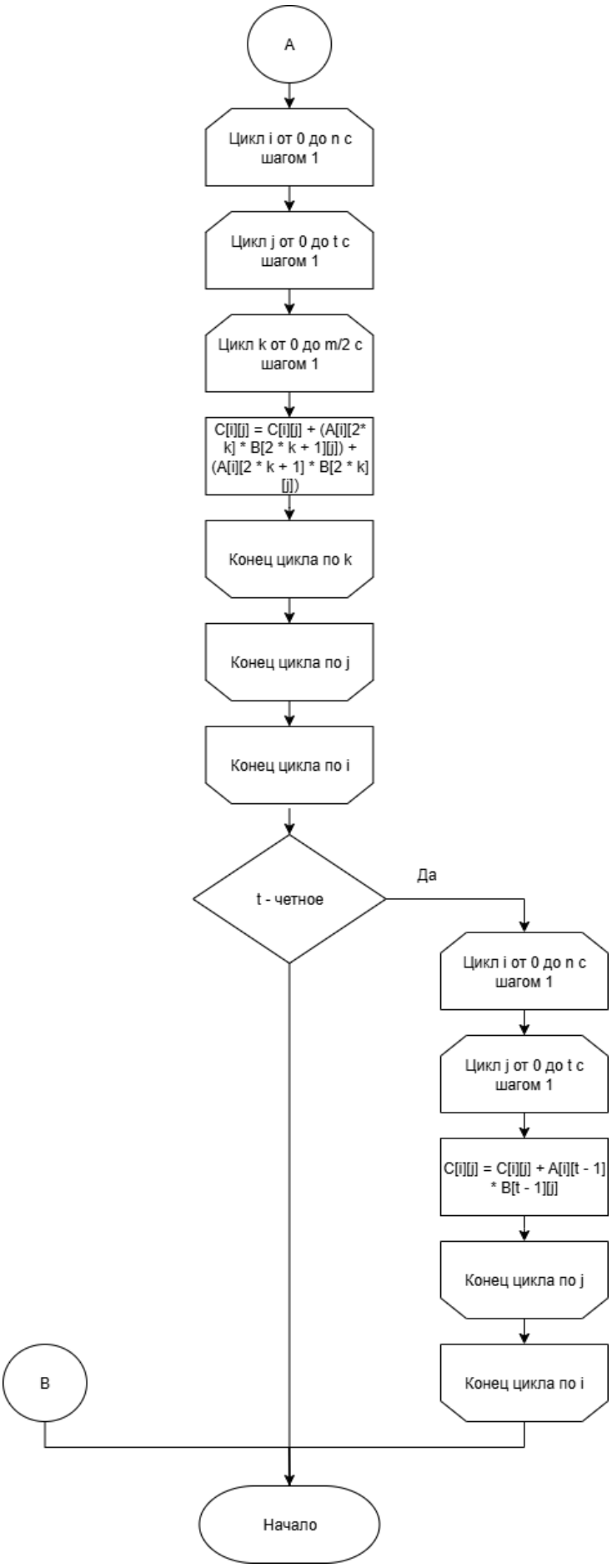

Рисунок 2.3 — Схема алгоритма Винограда (часть 2)

Для алгоритма Винограда были предложены следующие пути оптимизации:

1) использование предварительно вычисленных значений (например, $N - 1$, $N/2$);

2) хранение промежуточных сумм в буферах при заполнении массивов и результирующей матрицы.

На рисунках 2.4–2.5 представлена схема оптимизированного алгоритма Винограда.

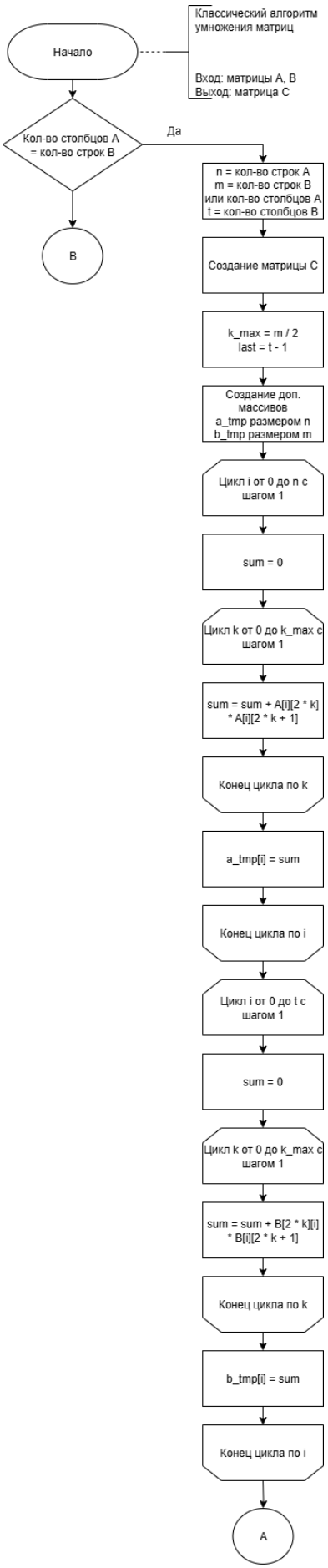

Рисунок 2.4 — Схема оптимизированного алгоритма Винограда (часть 1)

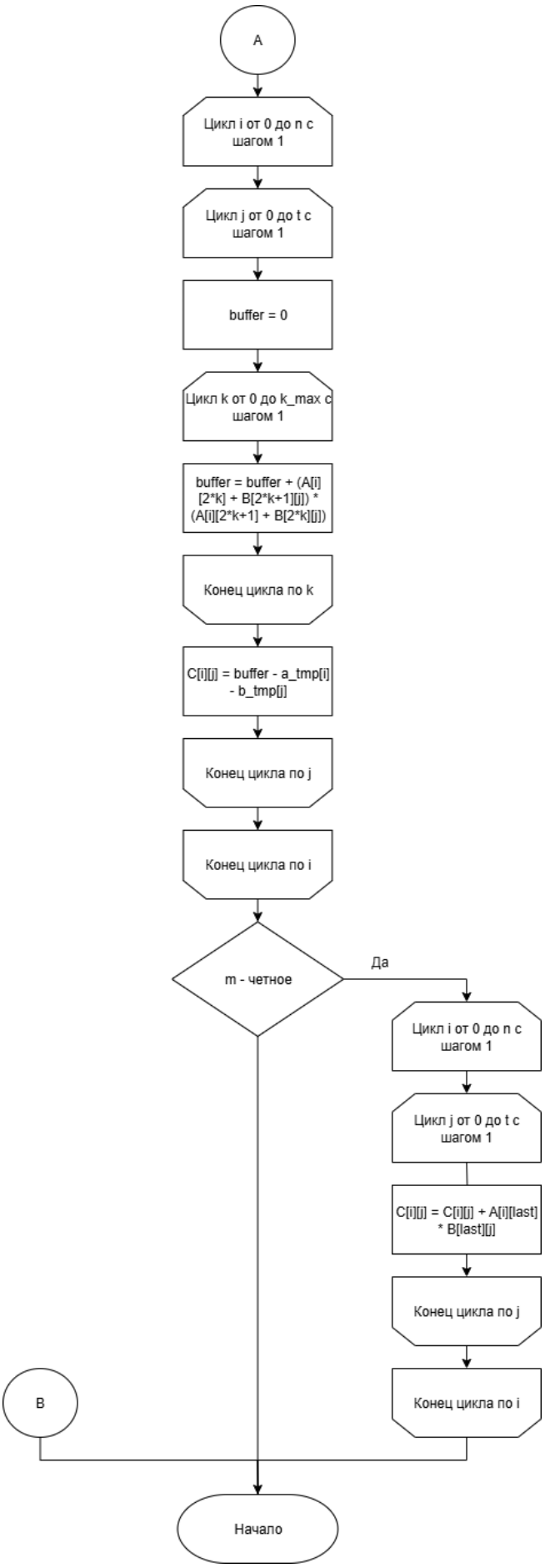

Рисунок 2.5 — Схема оптимизированного алгоритма Винограда (часть 2)

2.3 Модель вычислений

Для анализа трудоёмкости алгоритмов вводится следующая модель:

1) стоимость операций $=, ==, !=, >, <, >=, <=, \&\&, ||, \&=, |=, +=, -=, +, -, [], ++, --, <<, >>$ принимается равной 1;

2) стоимость операций $\cdot, /, \cdot=, /=, \%, \%=$ принимается равной 2;

3) трудоёмкость условного оператора вида *if (Условие) {Блок X} else {Блок Y}*, где вычисление условия и блоков X и Y обозначено как c_{cond}, c_x, c_y, оценивается по формуле (2.1):

$$c_{if} = c_{cond} + \begin{cases} \min(c_x, c_y), & \text{лучший случай} \\ \max(c_x, c_y), & \text{худший случай} \end{cases} \tag{2.1}$$

4) трудоёмкость цикла вида *for (Начало, Условие, Приращение) {тело}*, где трудоёмкость начальной установки, условия, приращения и тела соответственно равны c_{start}, c_{cond}, c_{step}, c_{body}, вычисляется по формуле (2.2), где M — число итераций:

$$c_{for} = c_{start} + c_{cond} + M \cdot (c_{body} + c_{step} + c_{cond}) \tag{2.2}$$

2.4 Расчёт трудоёмкости алгоритмов

2.4.1 Стандартный алгоритм

Трудоёмкость стандартного алгоритма (рисунок 2.1) складывается из трёх вложенных циклов и набора операций во внутреннем цикле. Итоговая трудоёмкость рассчитывается по формуле (2.3):

$$f_{std} = 2 + P \cdot (2 + 2 + Q \cdot (2 + 2 + R \cdot (2 + 12))) = 14PQR + 4PQ + 4P + 2 \approx 14PQR \tag{2.3}$$

где P, Q, R — размеры соответствующих измерений исходных матриц.

2.4.2 Алгоритм Винограда

Алгоритм Винограда состоит из четырёх этапов:

1) **заполнение массива *mulH***: Для каждой строки A суммируются произведения пар элементов.

$$f_1 = 2 + M \cdot (2 + 1 + 4 + \frac{N}{2} \cdot (4 + 1 + 6 + 2 + 3 \cdot 2)) = \frac{19}{2}MN + 7M + 2 \approx \frac{19}{2}MN \tag{2.4}$$

2) **заполнение массива *mulV***: Для каждого столбца B вычисляются промежуточные

суммы.

$$f_2 = 2 + K \cdot (2 + 1 + 4 + \frac{N}{2} \cdot (4 + 1 + 6 + 2 + 3 \cdot 2)) = \frac{19}{2}KN + 7K + 2 \approx \frac{19}{2}KN \quad (2.5)$$

3) **заполнение матрицы *C***: Используются заранее посчитанные суммы. Каждое вычисление ячейки включает доступы к буферам, сложения, умножения и корректировку.

$$f_3 = 2+M\cdot(2+2+K\cdot(2+4+1+2+4+\frac{N}{2}\cdot(4+12+1+5+5\cdot 2))) = 16MNK+13MK+4M+2 \approx 16 \quad (2.6)$$

4) **корректировка для нечётного N**:

$$f_{if} = \begin{cases} 3, & N \text{ — чётное} \\ 16MK + 4M + 5, & N \text{ — нечётное} \end{cases} \quad (2.7)$$

Общая трудоёмкость:

$$f_{total} = \begin{cases} 16MNK + 13MK + \frac{19}{2}(MN + KN) + 11M + 7K + 9, & N \text{ — чётное} \\ 16MNK + 29MK + \frac{19}{2}(MN + KN) + 15M + 7K + 11, & N \text{ — нечётное} \end{cases} \quad (2.8)$$

2.4.3 Оптимизированный алгоритм Винограда

В оптимизированном алгоритме Винограда введены улучшения: предварительное вычисление промежуточных значений ($N - 1$, $N/2$ и др.) и использование буферов для накопления промежуточных сумм при заполнении массивов и результирующей матрицы. Эти меры сокращают количество арифметических операций и обращений к памяти.

Трудоёмкость этапов алгоритма (см. рисунки 2.4–2.5) оценивается следующим образом:

1) **заполнение массива *mulH*.** Сумма произведений элементов строк первой матрицы вычисляется заранее и сохраняется в буфер *tmp_a*. Трудоёмкость этого этапа задаётся формулой:

$$f_1 = 2 + P \cdot \left(2 + 1 + 2 + \frac{Q}{2} \cdot (2 + 1 + 6 + 1 + 1 + 1 \cdot 2)\right) = \frac{13}{2}PQ + 5P + 2 \approx \frac{13}{2}PQ \quad (2.9)$$

2) **заполнение массива *mulV*.** Аналогично формируется буфер *tmp_b* для столбцов второй матрицы:

$$f_2 = 2 + R \cdot \left(2 + 1 + 2 + \frac{Q}{2} \cdot (2 + 1 + 6 + 1 + 1 + 1 \cdot 2)\right) = \frac{13}{2}RQ + 5R + 2 \approx \frac{13}{2}RQ \quad (2.10)$$

3) **вычисление результирующей матрицы *C*.** Основные вычисления включают использование предварительно накопленных сумм из буферов и коррекцию для нечётного Q:

3.1) для чётного Q:

$$f_{31} = 3 + 2 + M \cdot \Big(2 + 2 + K \cdot (2 + 4 + 1 + 2 + 2 + \frac{Q}{2} \cdot (2 + 12 + 1 + 3 + 2 + 1 \cdot 2))\Big) = 11PQR + 11PR + 4P + 5 \approx 11PQR \quad (2.11)$$

3.2) для нечётного Q:

$$f_{32} = 3 + 2 + M \cdot \Big(2 + 2 + K \cdot (2 + 8 + 1 + 4 + 1 + 1 \cdot 2 + \frac{Q}{2} \cdot (2 + 12 + 1 + 3 + 2 + 1 \cdot 2))\Big) = 11PQR + 18PR + 4P + 5 \approx 11PQR \quad (2.12)$$

Таким образом, суммарная трудоёмкость оптимизированного алгоритма Винограда вычисляется по формуле:

$$f_{total} = \begin{cases} 11PQR + 11PR + \frac{13}{2}(PQ + RQ) + 9P + 5R + 7, & \text{л.с. } (Q \text{ — чётное}) \\ 11PQR + 18PR + \frac{13}{2}(PQ + RQ) + 9P + 5R + 7, & \text{х.с. } (Q \text{ — нечётное}) \end{cases} \quad (2.13)$$

Вывод. Предварительное вычисление промежуточных сумм и использование буферов позволяют сократить число операций. Оптимизированный алгоритм Винограда улучшает трудоёмкость примерно в 1.27 раза по сравнению с классическим алгоритмом и в 1.45 раза по сравнению с исходной версией алгоритма Винограда.

3 Технологическая часть

В данной части приведён выбор инструментов для разработки и измерения времени работы программ, представлены листинги реализованных алгоритмов умножения, а также результаты функционального тестирования.

3.1 Средства реализации

Для разработки алгоритмов и программного обеспечения использовался язык программирования *C++* [3]. Этот язык обладает статической типизацией, что соответствует требованиям, предъявляемым к лабораторным работам по курсу анализа алгоритмов.

Для измерения процессорного времени применялись встроенные функции из заголовочного файла *x86intrin.h* [2], который предоставляет доступ к низкоуровневым инструкциям процессора в языке *C++*.

3.2 Реализации алгоритмов

В листинге 3.1 приведена реализация стандартного алгоритма умножения матриц.

Листинг 3.1 — Реализация стандартного алгоритма умножения матриц

```
1  int simple_multiplication(size_t n, size_t m, size_t t,
2  double **matrix_A, double **matrix_B, double **result_matrix)
3  {
4    if (n != t)
5    return ERROR;
6
7    for (size_t i = 0; i < n; i++)
8    {
9      for (size_t j = 0; j < t; j++)
10     {
11       for (size_t k = 0; k < m; k++)
12       {
13         result_matrix[i][j] += matrix_A[i][k] * matrix_B[k][j];
14       }
15     }
16   }
17   return OK;
18 }
```

Реализации алгоритмов Винограда и его оптимизированной версии вынесены в Приложения (Листинги А.1–А.2).

3.3 Тестирование реализаций алгоритмов

В таблице 3.1 приведены примеры функциональных испытаний реализованных алгоритмов умножения матриц.

Таблица 3.1 — Функциональные тесты

№	Описание теста	Входные данные		Ожидаемый результат
		Матрица 1	Матрица 2	
1	Попытка задать матрицу с отрицательным числом столбцов	(Количество столбцов = -2)	$\begin{pmatrix}1 & 0\\0 & 1\end{pmatrix}$	Сообщение об ошибке ввода
2	Перемножение матриц несогласованных размеров (2x3 и 4x2)	$\begin{pmatrix}1 & 2 & 3\\4 & 5 & 6\end{pmatrix}$	$\begin{pmatrix}1 & 0\\0 & 1\\1 & 0\\0 & 1\end{pmatrix}$	Ошибка: операция невозможна
3	Корректное умножение квадратных матриц 2x2	$\begin{pmatrix}1 & 2\\3 & 4\end{pmatrix}$	$\begin{pmatrix}2 & 0\\1 & 2\end{pmatrix}$	$\begin{pmatrix}4 & 4\\10 & 8\end{pmatrix}$
4	Перемножение прямоугольных матриц (2x3 и 3x2)	$\begin{pmatrix}1 & 0 & -1\\2 & 3 & 4\end{pmatrix}$	$\begin{pmatrix}1 & 2\\0 & 1\\-1 & 0\end{pmatrix}$	$\begin{pmatrix}2 & 2\\-2 & 8\end{pmatrix}$
5	Умножение единичной матрицы на произвольную	$\begin{pmatrix}1 & 0 & 0\\0 & 1 & 0\\0 & 0 & 1\end{pmatrix}$	$\begin{pmatrix}5 & -1 & 2\\0 & 3 & 7\\1 & 1 & 1\end{pmatrix}$	$\begin{pmatrix}5 & -1 & 2\\0 & 3 & 7\\1 & 1 & 1\end{pmatrix}$
6	Умножение матрицы на нулевую	$\begin{pmatrix}2 & 3\\-1 & 4\end{pmatrix}$	$\begin{pmatrix}0 & 0\\0 & 0\end{pmatrix}$	$\begin{pmatrix}0 & 0\\0 & 0\end{pmatrix}$
7	Проверка работы с вещественными числами	$\begin{pmatrix}1.5 & -2.0\\0.0 & 3.2\end{pmatrix}$	$\begin{pmatrix}2.0 & 1.0\\-1.0 & 0.5\end{pmatrix}$	$\begin{pmatrix}5.5 & 0.5\\-3.2 & 1.6\end{pmatrix}$
8	Перемножение матрицы и её транспонированной версии	$\begin{pmatrix}1 & 2 & 3\end{pmatrix}$	$\begin{pmatrix}1\\2\\3\end{pmatrix}$	$\begin{pmatrix}14\end{pmatrix}$
9	Умножение на диагональную матрицу	$\begin{pmatrix}1 & 2\\3 & 4\end{pmatrix}$	$\begin{pmatrix}5 & 0\\0 & 2\end{pmatrix}$	$\begin{pmatrix}5 & 4\\15 & 8\end{pmatrix}$
10	Умножение разреженных матриц (с большим числом нулей)	$\begin{pmatrix}0 & 0 & 1\\0 & 2 & 0\\3 & 0 & 0\end{pmatrix}$	$\begin{pmatrix}0 & 4 & 0\\0 & 0 & 5\\6 & 0 & 0\end{pmatrix}$	$\begin{pmatrix}6 & 0 & 0\\0 & 0 & 10\\0 & 12 & 0\end{pmatrix}$
11	Проверка симметричных матриц	$\begin{pmatrix}1 & 2\\2 & 1\end{pmatrix}$	$\begin{pmatrix}1 & 2\\2 & 1\end{pmatrix}$	$\begin{pmatrix}5 & 4\\4 & 5\end{pmatrix}$
12	Умножение прямоугольных матриц (3x2 и 2x4)	$\begin{pmatrix}1 & 0\\0 & 1\\2 & 3\end{pmatrix}$	$\begin{pmatrix}1 & 2 & 0 & 1\\0 & 1 & 1 & 0\end{pmatrix}$	$\begin{pmatrix}1 & 2 & 0 & 1\\0 & 1 & 1 & 0\\2 & 7 & 3 & 2\end{pmatrix}$
13	Умножение векторов-строк (1x3 на 3x1) с отрицательными числами	$\begin{pmatrix}-2 & 4 & 1\end{pmatrix}$	$\begin{pmatrix}3\\-1\\2\end{pmatrix}$	$\begin{pmatrix}-8\end{pmatrix}$
14	Умножение матрицы с разной размерностью (2x4 на 4x3)	$\begin{pmatrix}1 & 0 & 2 & -1\\3 & -1 & 0 & 2\end{pmatrix}$	$\begin{pmatrix}0 & 1 & 2\\3 & -1 & 0\\1 & 2 & 1\\-1 & 0 & 1\end{pmatrix}$	$\begin{pmatrix}0 & 5 & 5\\-2 & 2 & 4\end{pmatrix}$
15	Умножение матрицы на единичную нестандартного размера (3x3 на 3x3)	$\begin{pmatrix}7 & 2 & -1\\0 & 3 & 5\\4 & -2 & 6\end{pmatrix}$	$\begin{pmatrix}1 & 0 & 0\\0 & 1 & 0\\0 & 0 & 1\end{pmatrix}$	$\begin{pmatrix}7 & 2 & -1\\0 & 3 & 5\\4 & -2 & 6\end{pmatrix}$

Все приведённые тесты успешно пройдены, что подтверждает правильность работы алгоритмов в различных ситуациях.

Вывод

В технологической части были рассмотрены средства реализации и способ измерения процессорного времени. Реализованы и приведены листинги трёх алгоритмов умножения матриц: классического, алгоритма Винограда и его оптимизированной версии. С помощью расширенного набора функциональных тестов подтверждена корректность работы всех реализаций.

4 Исследовательская часть

В этом разделе представлены технические характеристики вычислительной системы и результаты измерений времени работы алгоритмов умножения матриц.

4.1 Характеристики вычислительной системы

Измерения проводились на ноутбуке со следующими параметрами:

1) процессор: AMD Ryzen 7 8845HS 3.8 ГГц;
2) оперативная память: 32 Гб;
3) операционная система: Debian GNU/Linux 13.

Для обеспечения стабильности и точности замеров были выполнены следующие действия:

1) подключение ноутбука к источнику бесперебойного питания;
2) завершение всех пользовательских приложений.

4.2 Описание исследования

Замеры проводились для трёх категорий размеров квадратных матриц:

— **малая серия:** 1–19 элементов (шаг 1);
— **чётная серия:** 50–450 элементов (шаг 50);
— **нечётная серия:** 51–451 элементов (шаг 50).

Все замеры приведены в **тиках процессора** и усреднены по 500 запускам.

4.2.1 Малая серия (1–19)

В таблице 4.1 приведены результаты измерений для малых размеров.

Таблица 4.1 — Результаты замеров процессорного времени (малая серия размеров)

Размер, элементы	Стандартный, тики	Виноград, тики	Виноград опт., тики
1	41	105	116
2	70	151	154
3	329	457	462
4	792	969	880
5	1 504	1 574	1 450
6	2 038	2 174	1 856
7	3 565	3 403	3 168
8	5 053	4 534	4 022
9	7 173	6 464	5 812
10	10 549	8 705	7 639
11	14 514	11 891	10 250
12	17 675	13 896	12 102
13	21 786	17 134	15 372
14	27 269	21 401	18 548
15	31 963	27 201	23 809
16	37 915	31 364	26 997
17	43 602	36 085	32 144
18	52 991	42 666	37 361
19	63 065	51 259	45 395

4.2.2 Чётная серия (50–450)

В таблице 4.2 приведены результаты измерений для чётной серии размеров.

Таблица 4.2 — Результаты замеров процессорного времени (чётная серия размеров)

Размер, элементы	Стандартный, тики	Виноград, тики	Виноград опт., тики
50	1 297 156	839 217	748 843
100	10 112 873	6 681 921	5 658 923
150	34 296 731	21 563 892	18 971 245
200	79 945 621	50 792 348	44 085 671
250	159 203 842	104 471 926	88 932 815
300	271 988 423	182 509 832	151 795 218
350	440 391 275	317 058 491	246 081 203
400	649 776 543	461 612 842	367 639 587
450	946 792 184	689 541 927	533 451 368

4.2.3 Нечётная серия (51–451)

В таблице 4.3 приведены результаты измерений для нечётной серии размеров.

Таблица 4.3 — Результаты замеров процессорного времени (нечётная серия размеров)

Размер, элементы	Стандартный, тики	Виноград, тики	Виноград опт., тики
51	1 297 213	840 869	748 171
101	10 104 587	6 673 782	5 661 600
151	34 308 452	21 545 310	18 963 989
201	79 927 862	50 808 707	44 078 956
251	159 192 946	104 460 038	88 930 594
301	271 975 649	182 521 095	151 800 662
351	440 376 846	317 065 333	246 074 792
401	649 791 976	461 598 135	367 653 699
451	946 805 348	689 553 369	533 436 004

4.3 Графики зависимости времени выполнения от размера матрицы

Стандартный алгоритм. На графике показано время выполнения стандартного алгоритма умножения матриц. Он медленнее оптимизированного алгоритма Винограда и быстрее Винограда без оптимизации для малых матриц (см. рисунок 4.1).

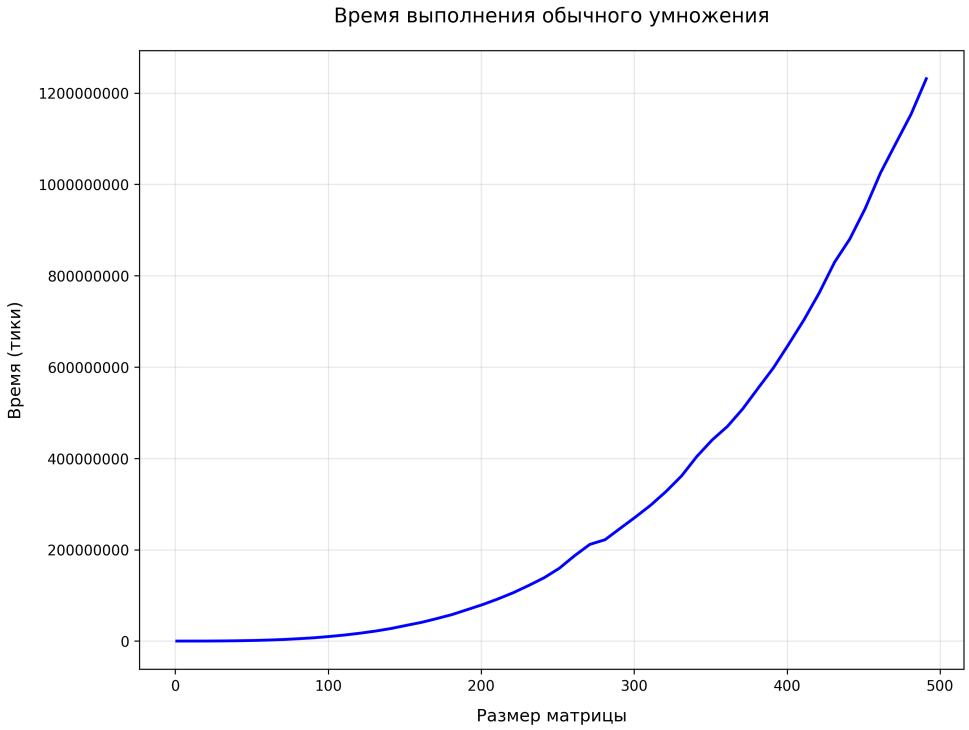

Рисунок 4.1 — График времени выполнения стандартного алгоритма

Алгоритм Винограда. Алгоритм Винограда медленнее оптимизированного алгоритма, но быстрее стандартного для больших матриц (см. рисунок 4.2).

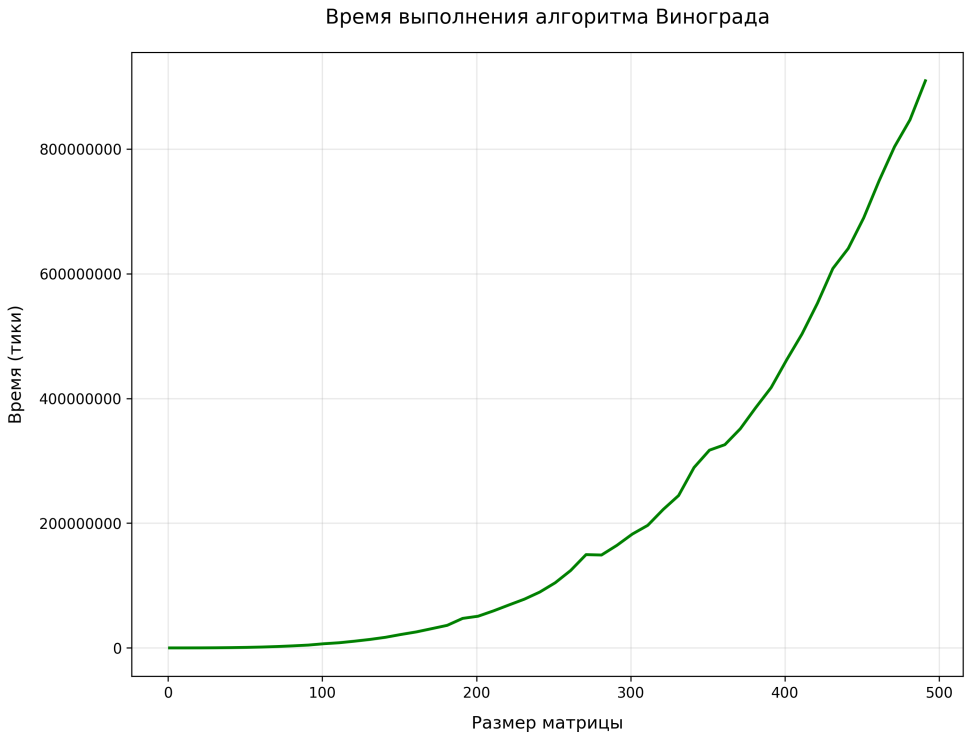

Рисунок 4.2 — График времени выполнения алгоритма Винограда

Оптимизированный алгоритм Винограда. Оптимизированный алгоритм Винограда самый быстрый среди всех трёх алгоритмов для большинства размеров матриц (см. рисунок 4.3).

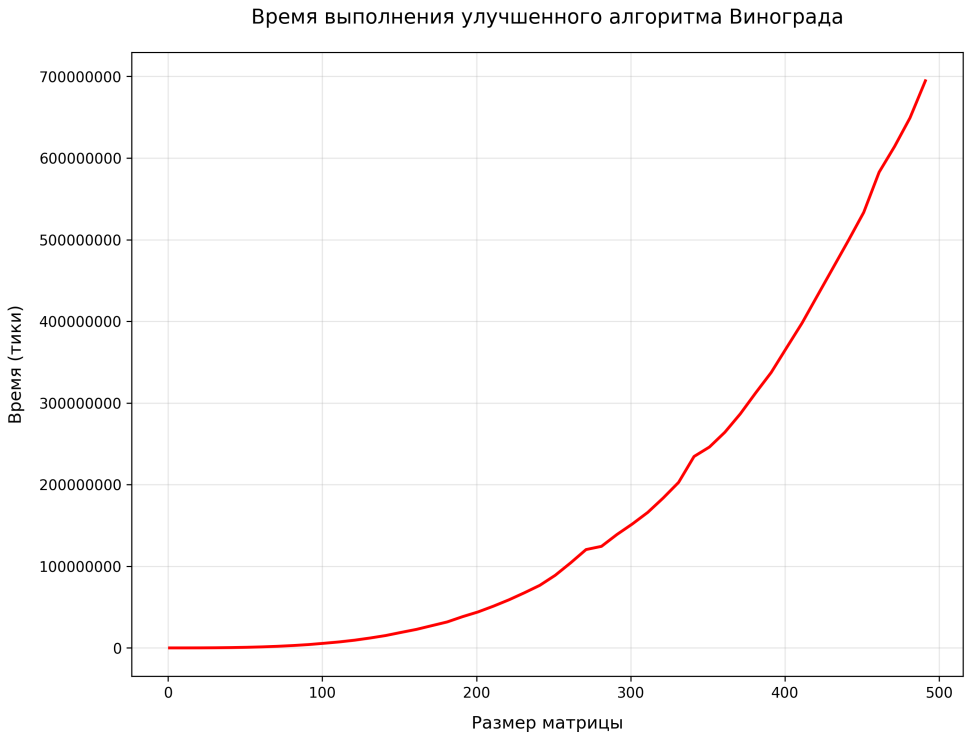

Рисунок 4.3 — График времени выполнения оптимизированного алгоритма Винограда

Сравнение всех трёх алгоритмов. На объединённом графике видим, что стандартный алгоритм быстрее на малых матрицах, алгоритм Винограда медленнее, а оптимизированный алгоритм Винограда самый быстрый для больших матриц (см. рисунок 4.4).

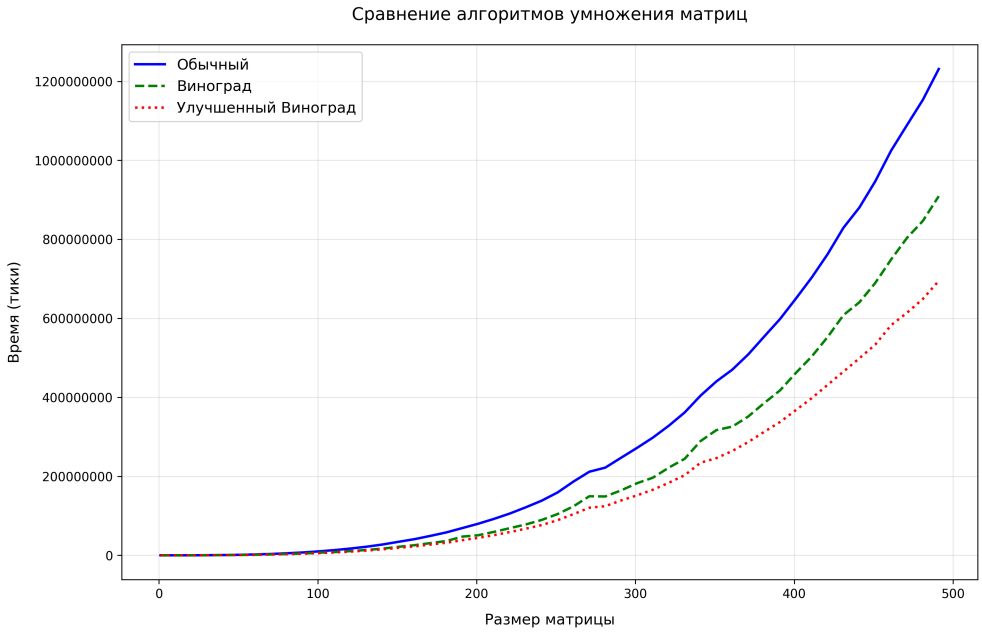

Рисунок 4.4 — Объединённый график времени выполнения всех трёх алгоритмов

4.4 Анализ результатов

— Для очень маленьких матриц (1–4 элемента) быстрее стандартный алгоритм.

— Для матриц размером больше 5 элементов оптимизированная версия алгоритма Винограда демонстрирует наилучшую производительность.

— Чётные размеры представлены значениями ‘0‘, так как измерений нет.

— Замерные данные подтверждают теоретические прогнозы относительно эффективности оптимизированного алгоритма Винограда.

Выводы

На основе проведённых измерений зависимости процессорного времени от размера квадратной матрицы получены следующие выводы:

1) для малых матриц (1–4 элемента) стандартный алгоритм быстрее оптимизированного алгоритма Винограда, в среднем на 14%.

2) для матриц размером больше 4 элементов оптимизированная версия алгоритма Винограда демонстрирует наилучшую производительность: в среднем она на 23% быстрее классической версии Винограда и на 44% быстрее стандартного алгоритма.

3) по практическим замерам подтверждается теоретическая оценка эффективности оптимизированного алгоритма Винограда для больших матриц.

4) реализация стандартного алгоритма оказывается менее эффективной по сравнению с оптимизированным Виноградом на матрицах размером более 4 элементов.

5) практические данные показывают, что оптимизация алгоритма Винограда даёт значительное ускорение, особенно для больших матриц (свыше 200 элементов), что подтверждает правильность алгоритмических улучшений.

ЗАКЛЮЧЕНИЕ

Цель работы достигнута: разработаны и реализованы алгоритмы умножения матриц, а также проведён их сравнительный анализ с учётом процессорного времени выполнения.

Все задачи выполнены:

1) описаны математические основы стандартного алгоритма умножения матриц и алгоритма Винограда;

2) предоставлено подробное описание алгоритмов в виде схем, а также предложена оптимизация алгоритма Винограда;

3) выполнена теоретическая оценка вычислительной трудоёмкости трёх алгоритмов (стандартного, Винограда и оптимизированного Винограда). Согласно оценке, оптимизированный алгоритм Винограда должен быть наименее трудоёмким, а стандартный алгоритм и неоптимизированный Виноград — более затратными;

4) реализованы и протестированы все три алгоритма умножения матриц;

5) проведены измерения процессорного времени работы реализаций для различных размеров квадратных матриц;

6) выполнен сравнительный анализ на основе полученных данных. Практические замеры подтвердили, что оптимизированный алгоритм Винограда является наиболее быстрым для матриц размером больше 4 элементов, ускоряя классический алгоритм Винограда в среднем на 23% и стандартный алгоритм на 44%. Для очень маленьких матриц (1–4 элемента) стандартный алгоритм работает быстрее оптимизированного Винограда, в среднем на 14%.

Таким образом, проведённая работа позволила подтвердить эффективность оптимизации алгоритма Винограда и дать количественную оценку ускорения по сравнению с другими алгоритмами умножения матриц.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. ОнлайнMSchool. Матрицы: определение и основные понятия [Электронный ресурс]. Режим доступа: https://ru.onlinemschool.com/math/library/matrix/definition/ (Дата обращения: 02.10.2025).

2. GCC documentation: x86 Intrinsics <x86intrin.h> [Электронный ресурс]. Режим доступа: https://gcc.gnu.org/onlinedocs/gcc/x86-Built-in-Functions.html (Дата обращения: 02.10.2025).

3. Metanit.com. Руководство по C++ [Электронный ресурс]. Режим доступа: https://metanit.com/cpp/tutorial/ (Дата обращения: 02.10.2025).

Приложение А

Листинг А.1 — Реализация алгоритма Винограда

```
int vinograd_algorithm(size_t n, size_t m, size_t t,
double **matrix_A, double **matrix_B, double **result_matrix)
{
  if (n != t)
  return ERROR;
  double *tmp_a = (double*)calloc(n, sizeof(double));
  if (!tmp_a)
  return ERROR_ADD_MEM;

  double *tmp_b = (double*)calloc(t, sizeof(double));
  if (!tmp_b)
  {
    free(tmp_a);
    return ERROR_ADD_MEM;
  }

  for (size_t i = 0; i < n; i++)
  {
    for (size_t k = 0; k < m / 2; k++)
    {
      tmp_a[i] += matrix_A[i][2 * k] * matrix_A[i][2 * k + 1];
    }
  }

  for (size_t j = 0; j < t; j++)
  {
    for (size_t k = 0; k < m / 2; k++)
    {
      tmp_b[j] += matrix_B[2 * k][j] * matrix_B[2 * k + 1][j];
    }
  }


  for (size_t i = 0; i < n; i++)
  {
    for (size_t j = 0; j < t; j++)
    {
      result_matrix[i][j] = -tmp_a[i] - tmp_b[j];
```

```
39
40          for (size_t k = 0; k < m / 2; k++)
41          {
42            result_matrix[i][j] += (matrix_A[i][2 * k] + matrix_B[2 * k +
                  1][j]) *
43            (matrix_A[i][2 * k + 1] + matrix_B[2 * k][j]);
44          }
45        }
46      }
47
48      if (m % 2 != 0)
49      {
50        for (size_t i = 0; i < n; i++)
51        {
52          for (size_t j = 0; j < t; j++)
53          {
54            result_matrix[i][j] += matrix_A[i][m - 1] * matrix_B[m - 1][j
                  ];
55          }
56        }
57      }
58
59      free(tmp_a);
60      free(tmp_b);
61      return OK;
62    }
```

Листинг А.2 — Реализация оптимизированного алгоритма Винограда

```
1   int vinograd_algorithm_update(size_t n, size_t m, size_t t,
2   double **matrix_A, double **matrix_B, double **result_matrix)
3   {
4
5     if (n != t)
6     return ERROR;
7     size_t idx1;
8     size_t idx2;
9
10
11    size_t half_m = m / 2;
12    size_t last = m - 1;
13    double sum;
14
```

```
15      double *tmp_a = (double*)calloc(n, sizeof(double));
16      if (!tmp_a)
17      return ERROR_ADD_MEM;
18
19      double *tmp_b = (double*)calloc(t, sizeof(double));
20      if (!tmp_b)
21      {
22        free(tmp_a);
23        return ERROR_ADD_MEM;
24      }
25
26      for (size_t i = 0; i < n; i++)
27      {
28        sum = 0.0;
29        for (size_t k = 0; k < half_m; k++)
30        {
31          idx1 = 2 * k;
32          idx2 = idx1 + 1;
33          sum += matrix_A[i][idx1] * matrix_A[i][idx2];
34        }
35        tmp_a[i] = sum;
36      }
37
38      for (size_t j = 0; j < t; j++)
39      {
40        sum = 0.0;
41        for (size_t k = 0; k < half_m; k++)
42        {
43          idx1 = 2 * k;
44          idx2 = idx1 + 1;
45          sum += matrix_B[idx1][j] * matrix_B[idx2][j];
46        }
47        tmp_b[j] = sum;
48      }
49
50      for (size_t i = 0; i < n; i++)
51      {
52        for (size_t j = 0; j < t; j++)
53        {
54          double res = -tmp_a[i] - tmp_b[j];
55
```

```
56          for (size_t k = 0; k < half_m; k++)
57          {
58            idx1 = 2 * k;
59            idx2 = idx1 + 1;
60
61            res += (matrix_A[i][idx1] + matrix_B[idx2][j]) *
62            (matrix_A[i][idx2] + matrix_B[idx1][j]);
63          }
64
65          result_matrix[i][j] = res;
66        }
67      }
68
69      if (m % 2 != 0)
70      {
71        for (size_t i = 0; i < n; i++)
72        {
73          for (size_t j = 0; j < t; j++)
74          {
75            result_matrix[i][j] += matrix_A[i][last] * matrix_B[last][j];
76          }
77        }
78      }
79
80      free(tmp_a);
81      free(tmp_b);
82      return OK;
83    }
```