СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цель работы:

1) Описать алгоритм умножения матриц.
2) Ввести модель вычислений.
3) Выполнить оценку трудоёмкости алгоритма умножения матриц в введённой модели вычислений.
4) Реализовать описанный алгоритм.
5) Описать архитектуру ЭВМ, для которой будут выполнены замеры произведения времени выполнения.
6) Можно ли выполнить алгоритм ЭВМ 4-го поколения? Что возможно выполнить? Есть ли у алгоритма характеристика времени?
7) Выполнить замеры, реализуя алгоритм для данных, являющихся худшими и лучшими случаями, по трудоёмкости алгоритма Винограда.
8) Проверить соответствие теоретической трудоёмкости алгоритма.
9) Провести сравнительный анализ реализации алгоритмов по критериям трудоёмкости.

1 Аналитическая часть

1.1 Алгоритмы умножения матриц

1.1.1 Стандартный метод

Стандартный алгоритм умножения матриц основывается на определении произведения матриц. Пусть даны матрицы A и B размерностей $m \times n$ и $n \times q$ соответственно. Тогда результатом умножения является матрица C размерности $m \times q$, элементы которой вычисляются по формуле:

$$c_{ij} = \sum_{k=1}^{n} a_{ik} \cdot b_{kj}, \qquad i = 1, \ldots, m,\ j = 1, \ldots, q. \tag{1.1}$$

Идея алгоритма заключается в последовательном вычислении каждого элемента результирующей матрицы C как скалярного произведения i-й строки матрицы A и j-го столбца матрицы B.

Вычислительная сложность стандартного алгоритма равна $O(m \cdot n \cdot q)$, так как для каждого из $m \cdot q$ элементов требуется n умножений и $(n - 1)$ сложений.

1.1.2 Метод Винограда

Метод Винограда является оптимизацией стандартного алгоритма и основан на сокращении числа арифметических операций за счёт предварительных вычислений. Рассмотрим скалярное произведение двух векторов длины n.

Для чётного n скалярное произведение можно представить в эквивалентной форме. Пусть $u = (u_1, \ldots, u_n)$ и $v = (v_1, \ldots, v_n)$. Тогда

$$u \cdot v = \sum_{k=1}^{n} u_k v_k \tag{1.2}$$

и эквивалентно

$$u \cdot v = \sum_{k=1}^{n/2} \big((u_{2k-1} + v_{2k})(u_{2k} + v_{2k-1}) - u_{2k-1}u_{2k} - v_{2k-1}v_{2k}\big). \tag{1.3}$$

Таким образом часть вычислений можно вынести за пределы основного цикла и сохранить промежуточные результаты:

— заранее вычисляются вспомогательные значения для строк матрицы A;

— аналогично, заранее вычисляются вспомогательные значения для столбцов матрицы B;

— затем, при вычислении каждого элемента результирующей матрицы, используются

подготовленные значения.

Для нечётного n вводят $t = \lfloor n/2 \rfloor$ и добавочный член с последними элементами:

$$u \cdot v = \sum_{k=1}^{t} \Big((u_{2k-1} + v_{2k})(u_{2k} + v_{2k-1}) - u_{2k-1}u_{2k} - v_{2k-1}v_{2k} \Big) + u_{2t+1}\, v_{2t+1}. \tag{1.4}$$

Для матриц это даёт следующую формулу для элемента c_{ij} (введём снова $t = \lfloor n/2 \rfloor$):

$$\begin{aligned} c_{ij} = \sum_{k=1}^{t} \Big((a_{i,2k-1} + b_{2k,j})(a_{i,2k} + b_{2k-1,j}) - a_{i,2k-1}a_{i,2k} - b_{2k-1,j}b_{2k,j} \Big) \\ + \begin{cases} a_{i,2t+1}\, b_{2t+1,j}, & \text{если } n \text{ нечётно}, \\ 0, & \text{если } n \text{ чётно}. \end{cases} \end{aligned} \tag{1.5}$$

Метод Винограда уменьшает количество операций умножения за счёт увеличения числа операций сложения, которые обычно дешевле по времени выполнения. Асимптотическая сложность метода также равна $O(m \cdot n \cdot q)$, однако на практике достигается выигрыш в константных множителях за счёт оптимизации числа умножений.

Вывод

В разделе рассмотрены два подхода к умножению матриц: стандартный метод и метод Винограда. Стандартный алгоритм прямолинеен и имеет вычислительную сложность $O(m \cdot n \cdot q)$, тогда как метод Винограда достигает практического ускорения за счёт предварительных вычислений, уменьшающих число умножений при обмене их на дополнительные сложения. Асимптотически обе схемы остаются эквивалентны по порядку сложности, но метод Винограда часто эффективнее на реальных машинах благодаря меньшему числу дорогих операций умножения, при этом его выгодность зависит от размера матриц и характера аппаратной реализации арифметики.

2 Конструкторская часть

2.1 Алгоритмы умножения матриц

Рисунок 2.1 — Визуализация схемы алгоритма стандартного умножения

Рисунок 2.2 — Визуализация схемы алгоритма Винограда

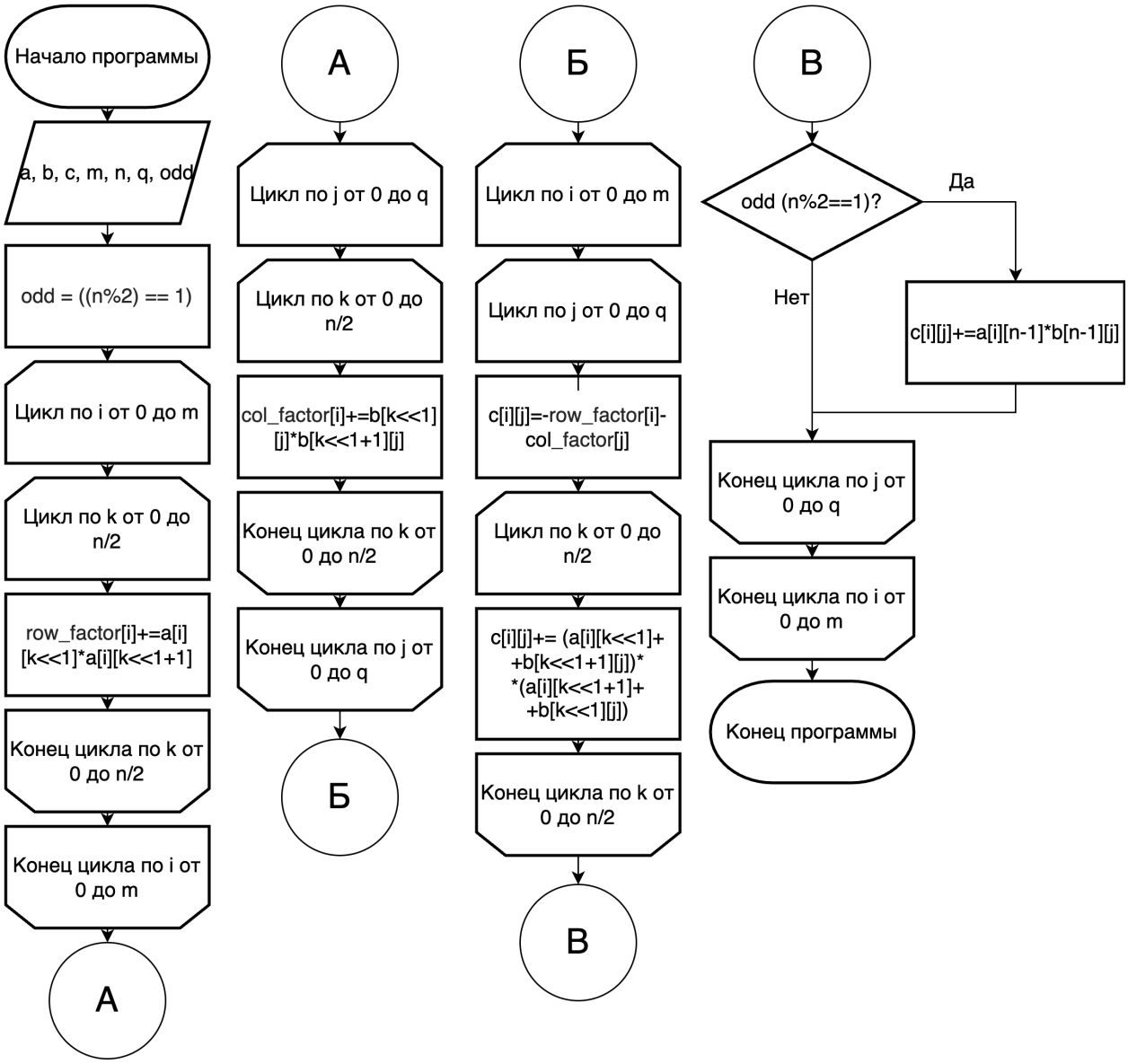

Рисунок 2.3 — Визуализация схемы алгоритма оптимизированного Винограда

2.2 Оценка трудоемкости

2.2.1 Модель вычислений

Для вычисления трудоемкости необходимо ввести модель вычислений:

1) операции из списка (2.1) имеют трудоемкость 1;

$$+, -, +=, -=, ==, !=, <, >, <=, >=, [], ++, --\tag{2.1}$$

2) операции из списка (2.2) имеют трудоемкость 2;

$$*, /, \%, *=, /=\tag{2.2}$$

3) трудоемкость оператора выбора `if условие then A else B` рассчитывается, как (2.3);

$$f_{if} = f_{\text{условия}} + \begin{cases} f_A, & \text{если условие выполняется,} \\ f_B, & \text{иначе.} \end{cases} \tag{2.3}$$

4) трудоемкость цикла рассчитывается, как (2.4);

$$f_{for} = f_{\text{инициализации}} + f_{\text{сравнения}} + N(f_{\text{тела}} + f_{\text{инкремента}} + f_{\text{сравнения}}) \tag{2.4}$$

5) трудоемкость вызова функции равна 0.

2.2.2 Стандартный алгоритм

$$f = 2 + m(2 + 2 + q(2 + 2 + n(2 + 1 + 2 + 2 + 2 + 1 + 1))) = 11mnq + 4mq + 4m + 2 \tag{2.5}$$

2.2.3 Алгоритм Винограда

Алгоритм Винограда, складывающийся из 4 циклов:

$$f1 = 2 + m(2 + 4 + \frac{n}{2}(4 + 1 + 1 + 4 + 2 + 5)) = \frac{17}{2}mn + 6m + 2 \tag{2.6}$$

$$f2 = 2 + q(2 + 4 + \frac{n}{2}(4 + 1 + 1 + 4 + 2 + 5)) = \frac{17}{2}qn + 6q + 2 \tag{2.7}$$

$$f3 = 2+m(2+2+q(2+7+4+\frac{n}{2}(4+2+1+4+1+5+2+5+4))) = 14mnq+13mq+4m+2 \tag{2.8}$$

$$f4 = 3 + \begin{cases} 0, \text{л.с. (n четное)} \\ 2 + m(2 + 2 + q(2 + 2 + 1 + 3 + 2 + 3)) = 13mq + 4m + 2, \text{х.с. (n нечетное)} \end{cases} \tag{2.9}$$

Итоговая трудоемкость - сумма трудоемкостей каждого цикла:

$$f = 14mnq + \frac{17}{2}mn + \frac{17}{2}qn + 6q + \begin{cases} 13mq + 10m + 9, \text{л.с. (n четное)} \\ 26mq + 14m + 11, \text{х.с. (n нечетное)} \end{cases} \tag{2.10}$$

2.2.4 Оптимизированный алгоритм Винограда

Оптимизированный алгоритм Винограда расчитывается аналогично:

$$f1 = 2 + m(2 + 4 + \frac{n}{2}(4 + 1 + 1 + 3 + 2 + 4)) = \frac{15}{2}mn + 6m + 2 \tag{2.11}$$

$$f2 = 2 + q(2 + 4 + \frac{n}{2}(4 + 1 + 1 + 3 + 2 + 4)) = \frac{15}{2}qn + 6q + 2 \tag{2.12}$$

$$\begin{aligned} f3 = 2 + m(2 + 2 + q(2 + 7 + 4 + \frac{n}{2}(4 + 2 + 1 + 3 + 1 + 4 + 2 + 4 + 3) + \\ \begin{cases} 1, \text{л.с. (n четное)} \\ 1 + 2 + 1 + 3 + 2 + 3, \text{х.с. (n нечетное)} \end{cases})) \\ = 12mnq + 4m + 2 + \begin{cases} 14mq, \text{л.с. (n четное)} \\ 25mq, \text{х.с. (n нечетное)} \end{cases} \end{aligned} \tag{2.13}$$

Итоговая трудоемкость - сумма трудоемкостей каждого цикла:

$$f = 12mnq + \frac{15}{2}mn + \frac{15}{2}qn + 10m + 6q + 6 + \begin{cases} 14mq, \text{л.с. (n четное)} \\ 25mq, \text{х.с. (n нечетное)} \end{cases} \tag{2.14}$$

Вывод

В связи с тем, что в алгоритме Винограда вместо того, чтобы увеличивать k сразу на 2, k увеличивается сначала на 1, а затем в индексации элементов происходит умножение k на 2 (сдвиг на 1), математически трудоёмкость алгоритма Винограда больше, чем у стандартного.

3 Технологическая часть

3.1 Выбор языка и среды программирования

Для реализации программного обеспечения были использованы следующие средства:

— среда разработки Visual Studio Code [1];

— язык разработки C++, компилятор gcc [2].

Выбор языка программирования C++ обусловлен наличием всех необходимых библиотек, обеспечивающих решение поставленных задач, а также статической типизацией. В качестве среды разработки используется Visual Studio Code, поскольку она предоставляет множество инструментов для написания и отладки программ на C++.

Используемые библиотеки:

— iostream [3]

— vector [4]

— chrono [5]

— random [6]

3.2 Код алгоритмов

Листинг 3.1 — Стандартный алгоритм умножения

```
/**
 * @brief стандартное умножение матриц
 */
matrix_t multiply_standard(const matrix_t &a, const matrix_t &b)
{
    int m = a.size();
    int n = a[0].size();
    int q = b[0].size();

    matrix_t c(m, vector<matrix_type_t>(q, 0));

    for (int i = 0; i < m; i++)
        for (int j = 0; j < q; j++)
            for (int k = 0; k < n; k++)
                c[i][j] += a[i][k] * b[k][j];

    return c;
}
```

Листинг 3.2 — Алгоритм Винограда

```
/**
 * @brief умножение матриц алгоритмом Винограда
 */
matrix_t multiply_winograd(const matrix_t &a, const matrix_t &b)
{
    int m = a.size();
    int n = a[0].size();
    int q = b[0].size();

    matrix_t c(m, vector<matrix_type_t>(q, 0));

    vector<matrix_type_t> row_factor(m, 0);
    vector<matrix_type_t> col_factor(q, 0);

    for (int i = 0; i < m; i++)
        for (int k = 0; k < n / 2; k++)
            row_factor[i] += a[i][2 * k] * a[i][2 * k + 1];

    for (int j = 0; j < q; j++)
        for (int k = 0; k < n / 2; k++)
            col_factor[j] += b[2 * k][j] * b[2 * k + 1][j];

    for (int i = 0; i < m; i++)
        for (int j = 0; j < q; j++)
        {
            c[i][j] = -row_factor[i] - col_factor[j];
            for (int k = 0; k < n / 2; k++)
                c[i][j] += (a[i][2 * k] + b[2 * k + 1][j]) *
                           (a[i][2 * k + 1] + b[2 * k][j]);
        }

    if (n % 2 == 1)
        for (int i = 0; i < m; i++)
            for (int j = 0; j < q; j++)
                c[i][j] += a[i][n - 1] * b[n - 1][j];

    return c;
}
```

Листинг 3.3 — Оптимизированный алгоритм Винограда

```
/**
 * @brief умножение матриц оптимизированным алгоритмом Винограда
 */
matrix_t multiply_optimized_winograd(const matrix_t &a, const
    matrix_t &b)
{
    int m = a.size();
    int n = a[0].size();
    int q = b[0].size();

    bool odd = (n % 2 == 1);

    matrix_t c(m, vector<matrix_type_t>(q, 0));

    vector<matrix_type_t> row_factor(m, 0);
    vector<matrix_type_t> col_factor(q, 0);

    for (int i = 0; i < m; i++)
        for (int k = 0; k < n / 2; k++)
            row_factor[i] += a[i][(k << 1)] * a[i][(k << 1) + 1];

    for (int j = 0; j < q; j++)
        for (int k = 0; k < n / 2; k++)
            col_factor[j] += b[(k << 1)][j] * b[(k << 1) + 1][j];

    for (int i = 0; i < m; i++)
    {
        for (int j = 0; j < q; j++)
        {
            c[i][j] = -row_factor[i] - col_factor[j];
            for (int k = 0; k < n / 2; k++)
                c[i][j] += (a[i][(k << 1)] + b[(k << 1) + 1][j]) *
                           (a[i][(k << 1) + 1] + b[(k << 1)][j]);
            if (odd)
                c[i][j] += a[i][n - 1] * b[n - 1][j];
        }
    }

    return c;
}
```

4 Исследовательская часть

4.1 Характеристики устройства

Исследование проводилось на устройстве со следующими характеристиками:

— операционная система — macOS Sequoia версии 15.3.1 [7];

— центральный процессор (CPU) — Apple M3 Max, 16 ядер 16 (12 производительности и 4 эффективности) [8];

— графическая подсистема — встроенное в чип графическое ядро с 40 графическими ядрами (GPU) [8];

— объединённая память (UM) — 48ГБ объединённой памяти [8].

Объединённая память (Unified Memory), применяемая в архитектуре Apple Silicon, обеспечивает совместное использование памяти CPU и GPU, что снижает задержки при доступе к данным и повышает производительность.

4.2 Демонстрация работы программы

```
artem@MacBook-Pro-3 lab_01 % ./app.exe
Режимы работы:
1) Ручной режим
2) Режим массированного замера времени
Введите номер режима работы:
2

=== Лучший случай для алгоритма Винограда (четные размеры) ===
N = 100 (усреднено по 100 прогонам)
Стандартный: 5 ms
Виноград: 4 ms
Оптимизированный Виноград: 4 ms

N = 200 (усреднено по 100 прогонам)
Стандартный: 42 ms
Виноград: 34 ms
Оптимизированный Виноград: 34 ms

N = 300 (усреднено по 50 прогонам)
Стандартный: 144 ms
Виноград: 123 ms
Оптимизированный Виноград: 124 ms

N = 400 (усреднено по 50 прогонам)
Стандартный: 356 ms
Виноград: 293 ms
Оптимизированный Виноград: 293 ms

N = 500 (усреднено по 25 прогонам)
Стандартный: 692 ms
Виноград: 567 ms
Оптимизированный Виноград: 569 ms

=== Худший случай для алгоритма Винограда (нечетные размеры) ===
N = 101 (усреднено по 100 прогонам)
Стандартный: 5 ms
Виноград: 4 ms
Оптимизированный Виноград: 4 ms

N = 201 (усреднено по 50 прогонам)
Стандартный: 42 ms
Виноград: 35 ms
Оптимизированный Виноград: 35 ms

N = 301 (усреднено по 50 прогонам)
Стандартный: 143 ms
Виноград: 124 ms
Оптимизированный Виноград: 124 ms

N = 401 (усреднено по 25 прогонам)
Стандартный: 352 ms
Виноград: 294 ms
Оптимизированный Виноград: 294 ms

N = 501 (усреднено по 25 прогонам)
Стандартный: 686 ms
Виноград: 570 ms
Оптимизированный Виноград: 571 ms
```

Рисунок 4.1 — Демонстрация работы программы

4.3 Информация об исследовании

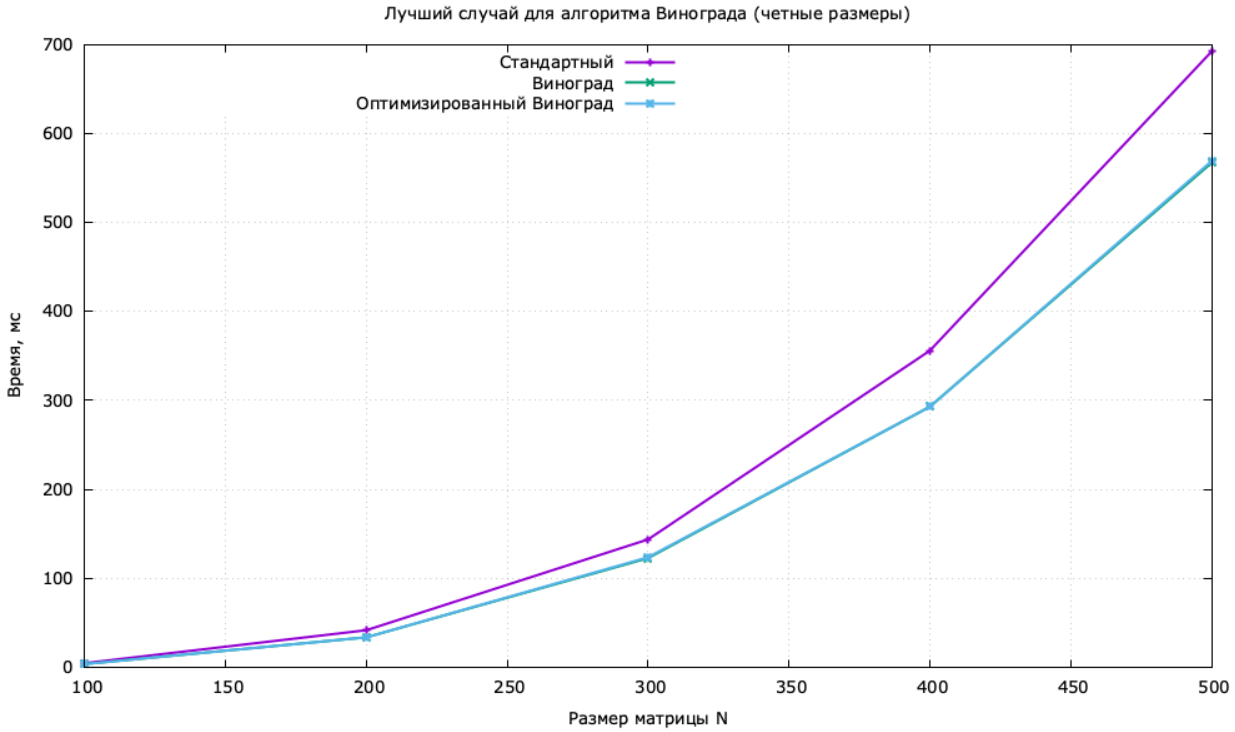

Рисунок 4.2 — Лучший случай для алгоритма Винограда

Таблица 4.1 — Лучший случай для алгоритма Винограда (четные размеры)

N	Стандартный, мс	Виноград, мс	Оптимизированный Виноград, мс
100	5	4	4
200	42	34	34
300	144	123	124
400	356	293	293
500	692	567	569

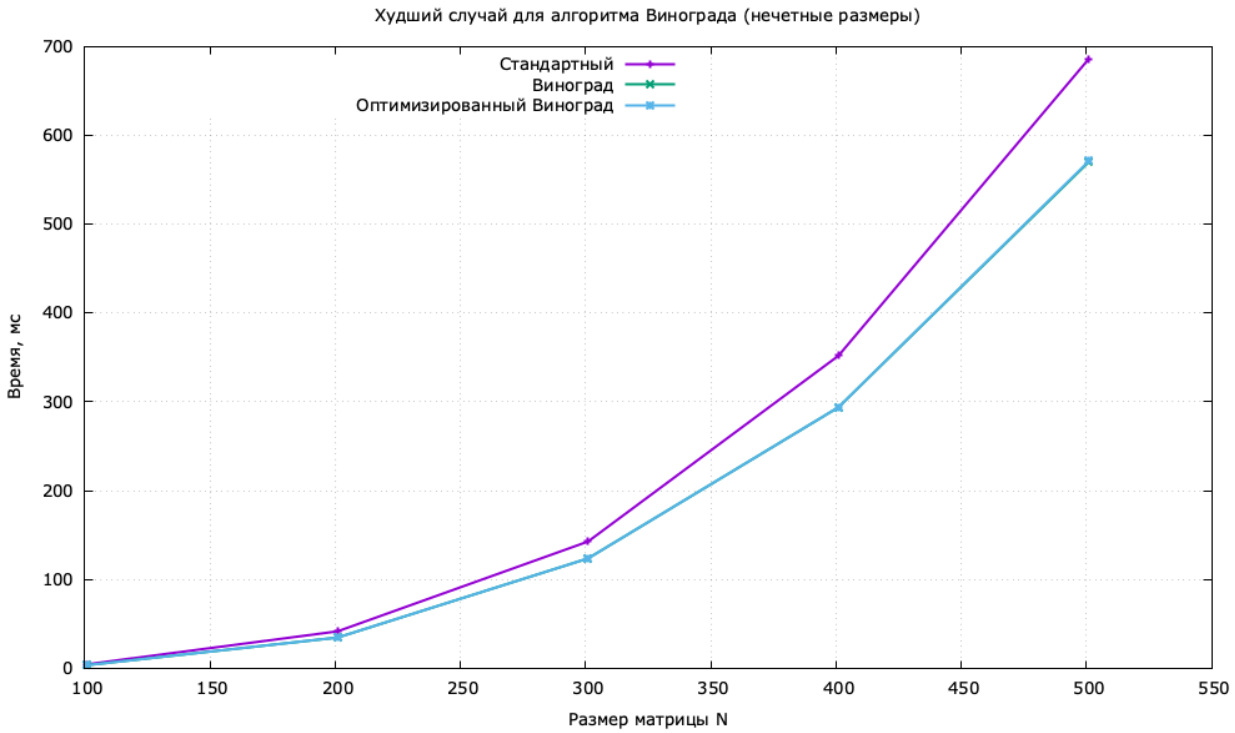

Рисунок 4.3 — Худший случай для алгоритма Винограда

Таблица 4.2 — Худший случай для алгоритма Винограда (нечетные размеры)

N	Стандартный, мс	Виноград, мс	Оптимизированный Виноград, мс
101	5	4	4
201	42	35	35
301	143	124	124
401	352	294	294
501	686	570	571

Выводы

На основании полученных в ходе исследования данных, следует что даже при указании компилятору не использовать оптимизацию (флаг -O0), он всё равно производит оптимизации, по этой причине реализованный алгоритм Винограда эффективнее, чем стандартный, несмотря на то, что математически должно быть обратное, а разницы между стандартным алгоритмом Винограда и его оптимизированной версией нет.

ЗАКЛЮЧЕНИЕ

В ходе выполнения лабораторной работы была проведена аналитическая, конструкторская и технологическая разработка алгоритмов умножения матриц. Рассмотрены два основных подхода: стандартный метод и алгоритм Винограда, включая его оптимизированную версию.

Аналитический анализ показал, что стандартный алгоритм имеет временную сложность $O(m \cdot n \cdot q)$, в то время как метод Винограда сохраняет ту же асимптотическую сложность, но снижает константные множители за счёт уменьшения количества операций умножения и увеличения числа сложений. Оптимизированный алгоритм Винограда минимизирует избыточные вычисления, сохраняя ту же асимптотическую оценку.

Расчёт трудоёмкости в принятой модели вычислений показал, что математически алгоритм Винограда оказывается более затратным, чем стандартный метод. Однако экспериментальные исследования, проведённые на устройстве с процессором Apple M3 Max и 48 ГБ объединённой памяти, выявили обратную картину. Даже при использовании флага компиляции `-O0`, который должен отключать оптимизации, компилятор применял их, и алгоритм Винограда оказался быстрее стандартного метода. При этом различий во времени выполнения между обычным и оптимизированным алгоритмом Винограда выявлено не было.

Технологическая реализация алгоритмов выполнена на языке C++ с использованием библиотек `iostream`, `vector`, `chrono` и `random`. Программа корректно обрабатывала матрицы различных размеров и подтвердила теоретические и экспериментальные выводы.

Вывод: асимптотически все алгоритмы эквивалентны, однако в модели вычислений стандартный метод имеет меньшую трудоёмкость. На практике же, в силу особенностей компилятора и архитектуры процессора, алгоритмы Винограда демонстрируют лучшее время выполнения. Таким образом, выбор оптимального метода зависит не только от математической модели, но и от конкретной аппаратно-программной реализации. Дальнейшие исследования могут быть направлены на параллельные и многопоточные реализации для повышения производительности.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Visual Studio Code [Электронный ресурс]. — Режим доступа URL: https://code.visualstudio.com (дата обращения 16.09.2025).
2. GCC 14.2.0 [Электронный ресурс]. — Режим доступа URL: https://gcc.gnu.org/onlinedocs/14.2.0/ (дата обращения 16.09.2025).
3. Библиотека iostream [Электронный ресурс]. — Режим доступа URL: https://cppreference.com/w/cpp/header/iostream.html (дата обращения 16.09.2025).
4. Библиотека vector [Электронный ресурс]. — Режим доступа URL: https://cppreference.com/w/cpp/header/vector.html (дата обращения 16.09.2025).
5. Библиотека chrono [Электронный ресурс]. — Режим доступа URL: https://cppreference.com/w/cpp/header/chrono.html (дата обращения 16.09.2025).
6. Библиотека random [Электронный ресурс]. — Режим доступа URL: https://cppreference.com/w/cpp/header/random.html (дата обращения 16.09.2025).
7. Apple Inc. macOS Sequoia // Apple. [Электронный ресурс]. — Режим доступа URL: https://www.apple.com/macos/macos-sequoia/ (дата обращения: 19.05.2025).
8. Apple Inc. Apple M3 chip // Apple. [Электронный ресурс]. — Режим доступа URL: https://www.apple.com/newsroom/2023/10/apple-unveils-m3-m3-pro-and-m3-max-the-most-advanced-chips-for-a-personal-computer/ (дата обращения: 16.09.2025).