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»

Отчет по лабораторной работе №2 по курсу "Анализ алгоритмов"

Тема Алгоритм Копперсмита-Винограда

Студент Морозов Д.В.

Группа ИУ7-52Б

Оценка (баллы) ______

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

Москва — 2022 г.

Оглавление

Введение

Умножение матриц является основным инструментом линейной алгебры и имеет многочисленные применения в математике, физике, программировании. Целью данной лабораторной работы является изучение и реализация стандартного алгоритма умножения матриц и алгоритма Винограда.

Задачами данной лабораторной являются:

— изучение и реализация стандартного алгоритма умножения матриц и алгоритма Винограда;

— оценка трудоёмкости алгоритмов;

— сравнение затрат алгоритмов по времени выполнения.

1 Аналитическая часть

В этом разделе представлены описания стандартного алгоритма умножения матриц и алгоритма Винограда.

1.1 Наивный алгоритм умножения матриц

Пусть даны две матрицы:

$$X_{nm} = \begin{pmatrix} x_{11} & x_{12} & \dots & x_{1k} \\ x_{21} & x_{22} & \dots & x_{2k} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \dots & x_{nk} \end{pmatrix}, Y_{mk} = \begin{pmatrix} y_{11} & y_{12} & \dots & y_{1k} \\ y_{21} & b_{22} & \dots & y_{2k} \\ \vdots & \vdots & \ddots & \vdots \\ y_{m1} & y_{m2} & \dots & y_{mk} \end{pmatrix}. \quad (1.1)$$

Тогда их произведением будет матрица:

$$Z_{nk} = \begin{pmatrix} z_{11} & z_{12} & \dots & z_{1k} \\ z_{21} & z_{22} & \dots & z_{2k} \\ \vdots & \vdots & \ddots & \vdots \\ z_{n1} & z_{n2} & \dots & z_{nk} \end{pmatrix}, \quad (1.2)$$

где

$$z_{ij} = \sum_{l=1}^{m} x_{il} y_{lj} \text{ для } i = \overline{1, n}; j = \overline{1, k}. \quad (1.3)$$

Наивный алгоритм реализует формулу (1.3).

1.2 Алгоритм Копперсмита-Винограда

Рассмотрим два вектора: $A = (a_1, \ldots, a_n)$ и $B = (b_1, \ldots, b_n)$. Их скалярное произведение равно:

$$A \cdot B = \sum_{i=1}^{n} a_i \cdot b_i \tag{1.4}$$

$$A \cdot B = \sum_{i=1}^{n/2} (a_{2i} + b_{2i+1})(a_{2i+1} + b_{2i}) + t \tag{1.5}$$

$$t = \sum_{i=1}^{n/2} a_i \cdot a_{i+1} + \sum_{i=1}^{n/2} b_i \cdot b_{i+1} \tag{1.6}$$

В выражениях (1.5) и (1.6) требуется большее число вычислений, чем в (1.4), но они позволяют произвести предварительную обработку. Произведения $a_i \cdot a_{i+1}$ и $b_i \cdot b_{i+1}$ для $i = \overline{1, n/2}$ можно вычислить заранее для каждой соответствующей строки и столбца, что позволит уменьшить число умножений [1].

2 Конструкторская часть

В этом разделе приведены схемы алгоритмов и выполнена оценка трудоёмкости.

2.1 Разработка алгоритмов

Предполагается, что на вход всех алгоритмов поступили матрицы верного размера.

На рисунке 2.1 приведена схема стандартного алгоритма умножения матриц.

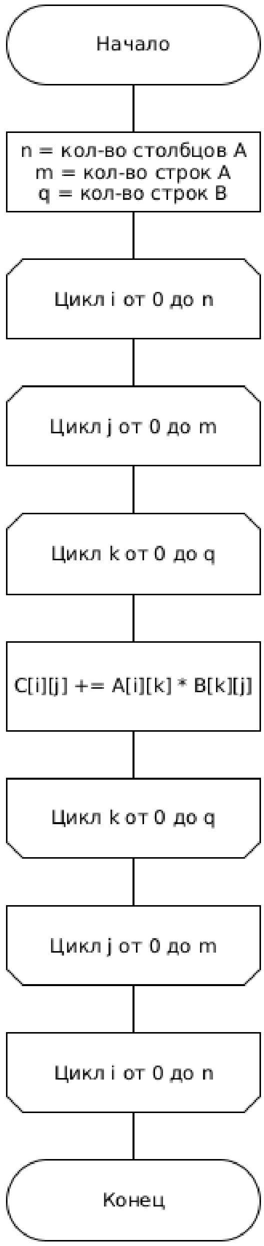

Рисунок 2.1 – Схема стандартного алгоритма умножения матриц

На рисунке 2.3 приведена схема алгоритма умножения матриц по Винограду.

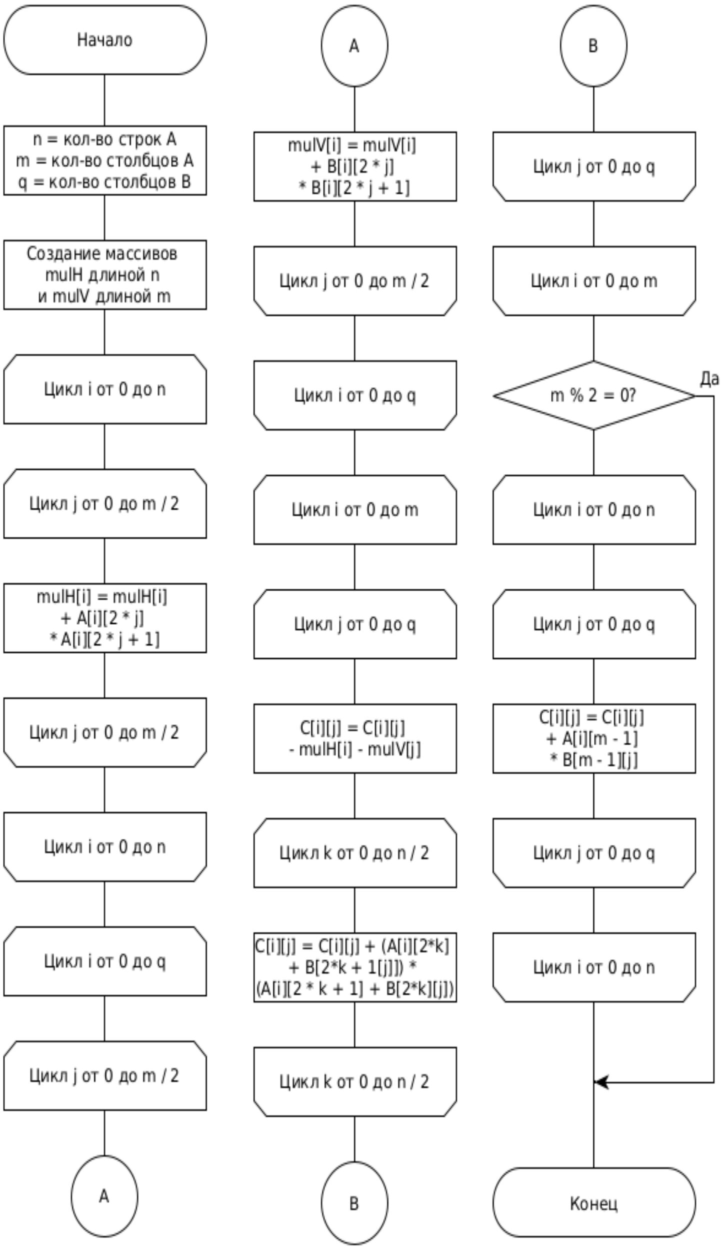

Рисунок 2.2 – Схема алгоритма умножения матриц по Винограду

На рисунке 2.3 приведена схема алгоритма умножения матриц по Винограду с оптимизациями.

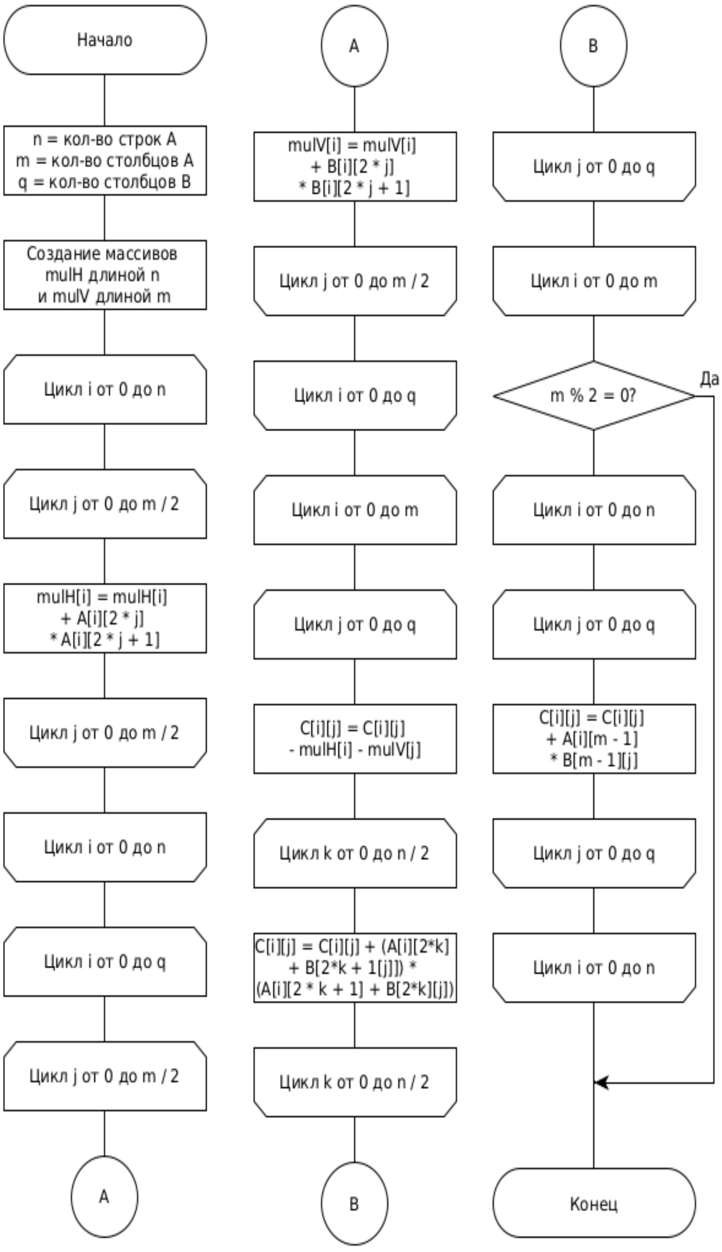

Рисунок 2.3 – Схема алгоритма умножения матриц по Винограду с оптимизациями

2.2 Оценка трудоемкости алгоритмов

2.2.1 Модель вычислений

Для оценки трудоёмкости алгоритмов введём модель вычислений.

1) Операции из списка (2.1) имеют трудоемкость 1.

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

2) Трудоемкость оператора выбора if условие then A else B рассчитывается, как (2.2).

$$f_{if} = f_{\text{условия}} + \begin{cases} f_A, & \text{если условие выполняется,} \\ f_B, & \text{иначе.} \end{cases} \tag{2.2}$$

3) Трудоемкость цикла рассчитывается, как (2.3).

$$\begin{gathered} f_{\text{цикла}} = f_{\text{инициализации}} + f_{\text{сравнения}} + \\ + N(f_{\text{тела}} + f_{\text{инкремента}} + f_{\text{сравнения}}) \end{gathered} \tag{2.3}$$

4) Трудоемкость вызова функции равна 0.

2.2.2 Стандартный алгоритм умножения матриц

Трудоёмкость стандартного алгоритма умножения матриц складывается из:

- трудомкости внешнего цикла по $i \in [1..M]$: $f = 2 + M \cdot (2 + f_{body})$;
- трудомкости цикла по $j \in [1..N]$: $f = 2 + N \cdot (2 + f_{body})$;
- трудомкости цикла по $k \in [1..Q]$: $f = 2 + 11Q$.

Общая трудоёмкость стандартного алгоритма умножения матриц:

$$f_{st} = 2 + M \cdot (4 + N \cdot (4 + 11Q)) = 2 + 4M + 4MN + 11MNQ, \tag{2.4}$$

$$f_{st} \approx 11MNQ. \tag{2.5}$$

2.2.3 Алгоритм Копперсмита-Винограда

Вычислим трудоёмкость алгоритма Копперсмита-Винограда:

1) трудоёмкость предобработки строк

$$\begin{gathered} f_{mulH} = \underbrace{1}_{=} + \underbrace{2}_{init} + M \cdot (\underbrace{2}_{inc} + \underbrace{4}_{init} + \frac{N}{2} \cdot \\ \cdot (\underbrace{4}_{inc} + \underbrace{6}_{[]} + \underbrace{2}_{+} + \underbrace{6}_{*} + \underbrace{1}_{=})), \end{gathered} \tag{2.6}$$

$$f_{mulH} = 3 + 6 \cdot M + 9.5 \cdot MN; \tag{2.7}$$

2) трудоёмкость предобработки столбцов (аналогично строкам)

$$f_{mulV} = 3 + 6 \cdot N + 9.5 \cdot MN; \tag{2.8}$$

3) трудоёмкость внутреннего цикла

$$f_{in} = \underbrace{4}_{init} + \frac{M}{2} \cdot (\underbrace{4}_{inc} + \underbrace{1}_{=} + \underbrace{12}_{[]} + \underbrace{5}_{+} + \underbrace{8}_{*}), \tag{2.9}$$

$$f_{in} = 4 + 15 \cdot M; \tag{2.10}$$

4) трудоёмкость внешнего цикла

$$\begin{gathered} f_{out} = \underbrace{2}_{init} + N \cdot (\underbrace{2}_{inc} + \underbrace{2}_{init} + Q \cdot \\ \cdot (\underbrace{2}_{inc} + \underbrace{1}_{=} + \underbrace{4}_{[]} + \underbrace{2}_{+} + f_{in})), \end{gathered} \tag{2.11}$$

$$f_{out} = 2 + 4 \cdot N + 13 \cdot NQ + 15 \cdot MNQ; \tag{2.12}$$

5) дополнительная трудоёмкость для худшего случая (кол-во столбцов 1-й матрицы нечётно)

$$\begin{gathered} f_{add} = \underbrace{2}_{init} + N \cdot (\underbrace{2}_{inc} + \underbrace{2}_{init} + Q \cdot \\ \cdot (\underbrace{2}_{inc} + \underbrace{1}_{=} + \underbrace{8}_{[]} + \underbrace{3}_{+} + \underbrace{2}_{*})), \end{gathered} \tag{2.13}$$

$$f_{add} = 2 + 4N + 16NQ. \tag{2.14}$$

Таким образом, трудоёмкость всего алгоритма в лучшем случае (кол-во столбцов 1-й матрицы чётно):

$$f_{best} = f_{mulH} + f_{mulV} + f_{out}, \tag{2.15}$$

$$f_{best} = 8 + 6M + 10N + 19MN + 13NQ + 15MNQ. \tag{2.16}$$

Трудоёмкость всего алгоритма в худшем случае:

$$f_{worst} = f_{best} + f_{add}, \tag{2.17}$$

$$f_{worst} = 10 + 6M + 14N + 19MN + 29NQ + 15MNQ. \tag{2.18}$$

2.2.4 Оптимизированный алгоритм Копперсмита-Винограда

Алгоритм Копперсмита-Винограда можно оптимизировать следующим образом:

— заменить операцию x = x + k на x += k;

— заменить умножение на 2 на побитовый сдвиг;

— предвычислять некоторые слагаемые для алгоритма.

Вычислим трудоёмкость оптимизированного алгоритма Копперсмита-Винограда:

1) трудоёмкость предобработки строк

$$f_{mulH} = \underbrace{1}_{=} + \underbrace{2}_{init} + M \cdot (\underbrace{2}_{inc} + \underbrace{2}_{init} + \frac{N}{2} \cdot \\ \cdot (\underbrace{2}_{inc} + \underbrace{1}_{-=} \underbrace{5}_{[]} + \underbrace{1}_{-} + \underbrace{2}_{*} +)), \tag{2.19}$$

$$f_{mulH} = 3 + 4 \cdot M + 5.5 \cdot MN; \tag{2.20}$$

2) трудоёмкость предобработки столбцов (аналогично строкам)

$$f_{mulV} = 3 + 4 \cdot N + 5.5 \cdot MN; \tag{2.21}$$

3) трудоёмкость внутреннего цикла

$$f_{in} = \underbrace{2}_{init} + \frac{N}{2} \cdot (\underbrace{2}_{inc} + \underbrace{1}_{+=} + \underbrace{8}_{[]} + \underbrace{4}_{+/-} + \underbrace{2}_{*}), \tag{2.22}$$

$$f_{in} = 2 + 8.5 \cdot N; \tag{2.23}$$

4) трудоёмкость внешнего цикла

$$f_{out} = \underbrace{2}_{init} + M \cdot (\underbrace{2}_{inc} + \underbrace{2}_{init} + Q \cdot \\ \cdot (\underbrace{2}_{inc} + \underbrace{2}_{=} + \underbrace{4}_{[]} + \underbrace{1}_{+} + f_{in})), \tag{2.24}$$

$$f_{out} = 2 + 4 \cdot M + 11 \cdot MQ + 8.5 \cdot MNQ; \tag{2.25}$$

5) дополнительная трудоёмкость для худшего случая (кол-во

столбцов 1-й матрицы нечётно)

$$f_{add} = \underbrace{2}_{init} + N \cdot (\underbrace{2}_{inc} + \underbrace{2}_{init} + Q \cdot \\ \cdot (\underbrace{2}_{inc} + \underbrace{1}_{=} + \underbrace{8}_{[]} + \underbrace{3}_{+} + \underbrace{2}_{*})), \tag{2.26}$$

$$f_{add} = 2 + 4N + 16NQ. \tag{2.27}$$

Таким образом, трудоёмкость всего алгоритма в лучшем случае (кол-во столбцов 1-й матрицы чётно):

$$f_{best} = f_{mulH} + f_{mulV} + f_{out}, \tag{2.28}$$

$$f_{best} = 8 + 6M + 10N + 19MN + 13NQ + 15MNQ. \tag{2.29}$$

Трудоёмкость всего алгоритма в худшем случае:

$$f_{worst} = f_{best} + f_{add}, \tag{2.30}$$

$$f_{worst} = 10 + 6M + 14N + 19MN + 29NQ + 15MNQ. \tag{2.31}$$

3 Технологическая часть

В данном разделе приведены требования к ПО, средства реализации и листинги кода.

3.1 Требования к ПО

Используемое ПО должно предоставлять возможность измерения процессорного времени.

3.2 Средства реализации

Для реализации данной лабораторной работы был выбран язык программирования C++ [2] и среда разработки CLion, которая позволяет замерять процессорное время с помощью пакета `<ctime>` [3].

3.3 Листинг кода

В листингах 3.1–3.9, приведены реализации алгоритмов.

Листинг 3.1 – Функция умножения матриц по стандартному алгоритму

```
static void mulStand(const Matrix& m1, const Matrix& m2,
    Matrix& res) {
    for (int i = 0; i < m1.rows; i++) {
        for (int j = 0; j < m1.cols; j++)
        for (int k = 0; k < m2.rows; k++)
        res.data[i][j] += m1.data[i][k] * m2.data[k][j];
    }
}
```

Листинг 3.2 – Функция умножения матриц по алгоритму Винограда (начало)

```
static void mulVinograd0(const Matrix& A, const Matrix& B,
    Matrix& res) {
    std::vector<int> mulH = A.preprocRows0();
    std::vector<int> mulV = B.preprocCols0();
    for (int i = 0; i < A.rows; i++) {
        for (int j = 0; j < B.cols; j++) {
            res.data[i][j] = -(mulH[i] + mulV[j]);
            for (int k = 0; k < A.cols/2; k++) {
                res.data[i][j] = res.data[i][j] +
                (A.data[i][k*2]+B.data[k*2+1][j])*
                (A.data[i][k*2+1]+B.data[k*2][j]);
            }
        }
    }
```

Листинг 3.3 – Функция умножения матриц по алгоритму Винограда (окончание)

```
    if (A.cols%2 != 0) {
        for (int i = 0; i < A.rows; i++) {
            for (int j = 0; j < B.cols; j++) {
                res.data[i][j] = res.data[i][j] +
                    A.data[i][A.cols-1]*B.data[A.cols-1][j];
            }
        }
    }
}
```

Листинг 3.4 – Функция предобработки строк

```
std::vector<int> preprocRows0() const {
    std::vector<int> res(rows);
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols/2; j++) {
            res[i] = res[i] + data[i][j*2] * data[i][j*2+1];
        }
    }
    return res;
}
```

Листинг 3.5 – Функция предобработки столбцов

```
std::vector<int> preprocCols0() const {
    std::vector<int> res(cols);
    for (int i = 0; i < cols; i++) {
        for (int j = 0; j < rows/2; j++) {
            res[i] = res[i] + data[j*2][i] * data[j*2+1][i];
        }
    }
    return res;
}
```

Листинг 3.6 – Функция умножения матриц по алгоритму Винограда с оптимизациями (начало)

```
static void mulVinograd3(const Matrix& A, const Matrix& B,
   Matrix& res) {
    std::vector<int> mulH = A.preprocRows3();
    std::vector<int> mulV = B.preprocCols3();
    for (int i = 0; i < A.rows; i++) {
        for (int j = 0; j < B.cols; j++) {
            int tmp = mulH[i] + mulV[j];
            for (int k = 1; k < A.cols; k += 2) {
                tmp += (A.data[i][k-1] + B.data[k][j]) *
                (A.data[i][k] + B.data[k-1][j]);
            }
            res.data[i][j] = tmp;
        }
    }
```

Листинг 3.7 – Функция умножения матриц по алгоритму Винограда с оптимизациями (окончание)

```
    if ((A.cols&1) != 0) {
        int n1 = A.cols - 1;
        for (int i = 0; i < A.rows; i++) {
            for (int j = 0; j < B.cols; j++) {
                res.data[i][j] += A.data[i][n1] *
                  B.data[n1][j];
            }
        }
    }
}
```

Листинг 3.8 – Функция предобработки строк с оптимизациями

```
std::vector<int> preprocRows3() const {
    std::vector<int> res(rows);
    for (int i = 0; i < rows; i++) {
        for (int j = 1; j < cols; j += 2) {
            res[i] -= data[i][j-1] * data[i][j];
        }
    }
    return res;
}
```

Листинг 3.9 – Функция предобработки столбцов с оптимизациями

```
std::vector<int> preprocCols3() const {
    std::vector<int> res(cols);
    for (int i = 0; i < cols; i++) {
        for (int j = 1; j < rows; j += 2) {
            res[i] -= data[j-1][i] * data[j][i];
        }
    }
    return res;
}
```

3.4 Функциональные тесты

В таблице 3.1 приведены тесты для функций, реализующих стандартный алгоритм умножения матриц, алгоритм Винограда и оптимизированный алгоритм Винограда. Все тесты пройдены успешно.

Таблица 3.1 – Тестирование функций

Матрица 1	Матрица 2	Ожидаемый результат
$\begin{pmatrix} 6 \end{pmatrix}$	$\begin{pmatrix} 7 \end{pmatrix}$	$\begin{pmatrix} 42 \end{pmatrix}$
$\begin{pmatrix} 1 & 2 & 3 \end{pmatrix}$	$\begin{pmatrix} 3 \\ 2 \\ 1 \end{pmatrix}$	$\begin{pmatrix} 10 \end{pmatrix}$
$\begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$	$\begin{pmatrix} 3 & 2 & 1 \end{pmatrix}$	$\begin{pmatrix} 3 & 2 & 1 \\ 6 & 4 & 2 \\ 9 & 6 & 3 \end{pmatrix}$
$\begin{pmatrix} 4 & 4 & 4 \\ 2 & 2 & 2 \\ 3 & 3 & 3 \end{pmatrix}$	$\begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$	$\begin{pmatrix} 12 \\ 6 \\ 9 \end{pmatrix}$
$\begin{pmatrix} 5 & 0 \\ 0 & 4 \end{pmatrix}$	$\begin{pmatrix} 0 & 7 \\ 8 & 0 \end{pmatrix}$	$\begin{pmatrix} 0 & 35 \\ 32 & 0 \end{pmatrix}$
$\begin{pmatrix} 2 & 3 \end{pmatrix}$	$\begin{pmatrix} 2 & 3 \end{pmatrix}$	Неверный размер

4 Исследовательская часть

В данном разделе произведено сравнение алгоритмов.

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялись замеры времени:

— операционная система — Ubuntu 22.04.1 Linux x86_64;

— оперативная память — 8 ГБ;

— процессор — AMD Ryzen 5 3550H [4].

Замеры проводились на ноутбуке, включенном в сеть электропитания. Во время замеров ноутбук не был нагружен сторонними приложениями.

4.2 Время выполнения алгоритмов

Замеры проводились для квадратных матриц одной размерности. На рисунке 4.1 представлен график, иллюстрирующий зависимость времени работы от размерности матриц для стандартного алгоритма и алгоритма Копперсмита-Винограда.

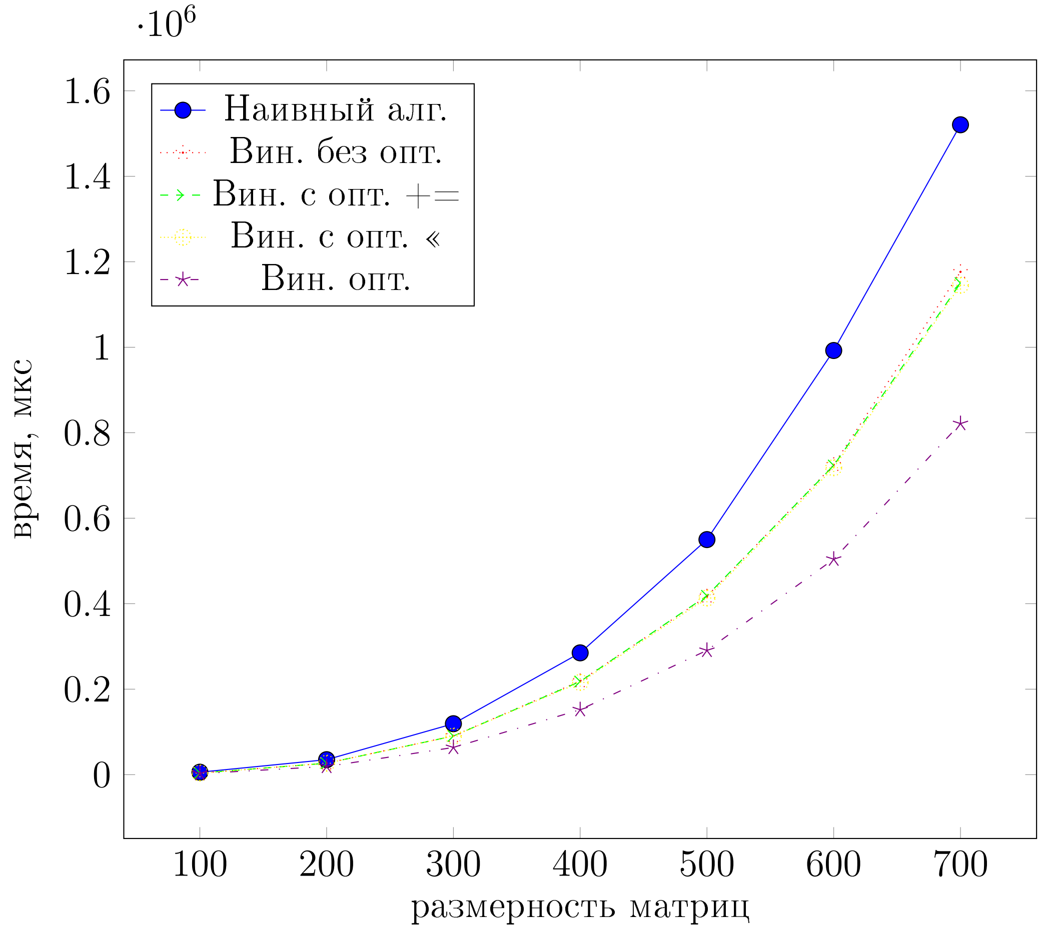

Рисунок 4.1 – Сравнение алгоритмов умножения матриц

Вывод

Алгоритм Копперсмита-Винограда превосходит по времени работы стандартный алгоритм на 85% (для матриц размерности $700 \cdot 700$).

Результаты замеров доказывают важность оптимизаций — версия алгоритма Копперсмита-Винограда с предварительными вычислениями быстрее обычной на 46% (для матриц размерности $300 \cdot 300$).

Заключение

Цель достигнута. В ходе выполнения лабораторной работы были решены следующие задачи:

- изучены и реализованы стандартный алгоритм умножения матриц и алгоритм Винограда;
- проведена оценка трудоёмкости алгоритмов;
- проведено сравнние алгоритмов по времени выполнения.

Алгоритм Копперсмита-Винограда превосходит по времени работы стандартный алгоритм на 85% (для матриц размерности $700 \cdot 700$).

Алгоритм Копперсмита-Винограда с оптимизациями быстрее алгоритма без оптимизаций на 46% (для матриц размерности $300 \cdot 300$).

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. 2018.

2. Документация по языку C++ | Microsoft Learn [Электронный ресурс]. Режим доступа: `https://learn.microsoft.com/ru-ru/cpp/cpp/?view=msvc-150` (дата обращения: 30.09.2022).

3. <ctime> | Microsoft Learn [Электронный ресурс]. Режим доступа: `https://learn.microsoft.com/en-us/cpp/standard-library/ctime?redirectedfrom=MSDN&view=msvc-160` (дата обращения: 30.09.2022).

4. AMD Ryzen™ 5 3550H [Электронный ресурс]. Режим доступа: `https://www.amd.com/en/products/apu/amd-ryzen-5-3550h` (дата обращения: 30.09.2022).