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»

на тему: «Алгоритмы умножения матриц»

Студент ИУ7-53Б (Группа) _____ (Подпись, дата) В. Марченко (И. О. Фамилия)

Преподаватель _____ (Подпись, дата) Ю. В. Строганов (И. О. Фамилия)

Преподаватель _____ (Подпись, дата) Л. Л. Волкова (И. О. Фамилия)

2022 г.

Оглавление

Введение

В настоящее время матричное исчисление широко применяется в различных областях математики, механики, теоретической физики, теоретической электротехники и т. д. [1]

Один из примеров использования матриц — аффинные преобразования в пространстве (или на плоскости) в компьютерной графике (параллельный перенос, поворот и масштабирование). Если объект нужно преобразовать, то нужно использовать операцию умножения матриц для всех вершин этого тела.

В математике умножение матриц — это бинарная операция, которая создает матрицу из двух исходных. Для умножения матриц количество столбцов в первой матрице должно быть равно количеству строк во второй. Результирующая матрица имеет количество строк первой и количество столбцов второй матрицы.

Помимо классического алгоритма умножения матриц, в данной лабораторной работе будут рассмотрены также алгоритм Копперсмита–Винограда и его оптимизированная версия.

1 Аналитическая часть

1.1 Цели и задачи

Цель работы: получить навыки оценки трудоемкости и временной эффективности на материале алгоритмов умножения матриц.

Задачи текущей лабораторной работы:

1) изучить и реализовать три алгоритма умножения матриц — классический, Копперсмита–Винограда и оптимизированный алгоритм Копперсмита–Винограда;

2) провести сравнительный анализ трудоемкости алгоритмов на основе теоретических расчетов и выбранной модели вычислений;

3) провести сравнительный анализ времени работы алгоритмов на основе экспериментальных данных.

1.2 Классический алгоритм

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

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix}, \tag{1.1}$$

$$B = \begin{bmatrix} b_{11} & b_{12} & \dots & b_{1r} \\ b_{21} & b_{22} & \dots & b_{2r} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \dots & b_{nr} \end{bmatrix}. \tag{1.2}$$

Произведение матрицы $A \equiv [a_{ij}]$ размера $m \times n$ на матрицу $B \equiv [b_{jk}]$ размера $n \times r$ есть матрица $C \equiv [c_{ik}]$ размера $m \times r$

$$C = AB \equiv [a_{ij}][b_{jk}] \equiv [c_{ik}], \tag{1.3}$$

где

$$c_{ik} = \sum_{j=1}^{n} a_{ij} b_{jk}. \tag{1.4}$$

Таким образом, элемент c_{ik} матрицы $C = AB$ есть сумма произведений элементов i-й строки матрицы A на соответствующие элементы k-го столбца матрицы B [2].

Операция умножения двух матриц выполнима только в том случае, если число столбцов в первом сомножителе равно числу строк во втором. В частности, умножение всегда выполнимо, если оба сомножителя — квадратные матрицы одного и того же порядка.

1.3 Алгоритм Копперсмита–Винограда

Если посмотреть на результат умножения двух матриц, то видно, что каждый элемент в нем представляет собой скалярное произведение соответствующих строки и столбца исходных матриц. Можно заметить также, что такое умножение допускает предварительную обработку, позволяющую часть работы выполнить заранее.

Рассмотрим два вектора $A = (a_1, a_2, a_3, a_4)$ и $B = (b_1, b_2, b_3, b_4)$. Их скалярное произведение равно:

$$A \cdot B = a_1 \cdot b_1 + a_2 \cdot b_2 + a_3 \cdot b_3 + a_4 \cdot b_4. \tag{1.5}$$

Это равенство можно переписать в виде:

$$A \cdot B = (a_1 + b_2)(a_2 + b_1) + (a_3 + b_4)(a_4 + b_3) - a_1 \cdot a_2 - a_3 \cdot a_4 - b_1 \cdot b_2 - b_3 \cdot b_4. \tag{1.6}$$

Выражение в правой части последнего равенства допускает предварительную обработку: его части можно вычислить заранее и запомнить для каждой строки первой матрицы и для каждого столбца второй.

1.4 Оптимизированный алгоритм Копперсмита–Винограда

Для оптимизации описанного в предыдущем пункте алгоритма Копперсмита–Винограда операция $x = x + k$ будет заменена на $x \mathrel{+}= k$, умножение на 2 будет заменено побитовым сдвигом влево, а некоторые слагаемые, используемые в алгоритме, будут вычисляться заранее.

Вывод из аналитической части

В текущем разделе были рассмотрены три алгоритма умножения матриц: классический, Копперсмита–Винограда и оптимизированный алгоритм Копперсмита–Винограда.

2 Конструкторская часть

2.1 Описание алгоритмов

На рисунках 2.1–2.5 показаны схемы трех алгоритмов умножения матриц.

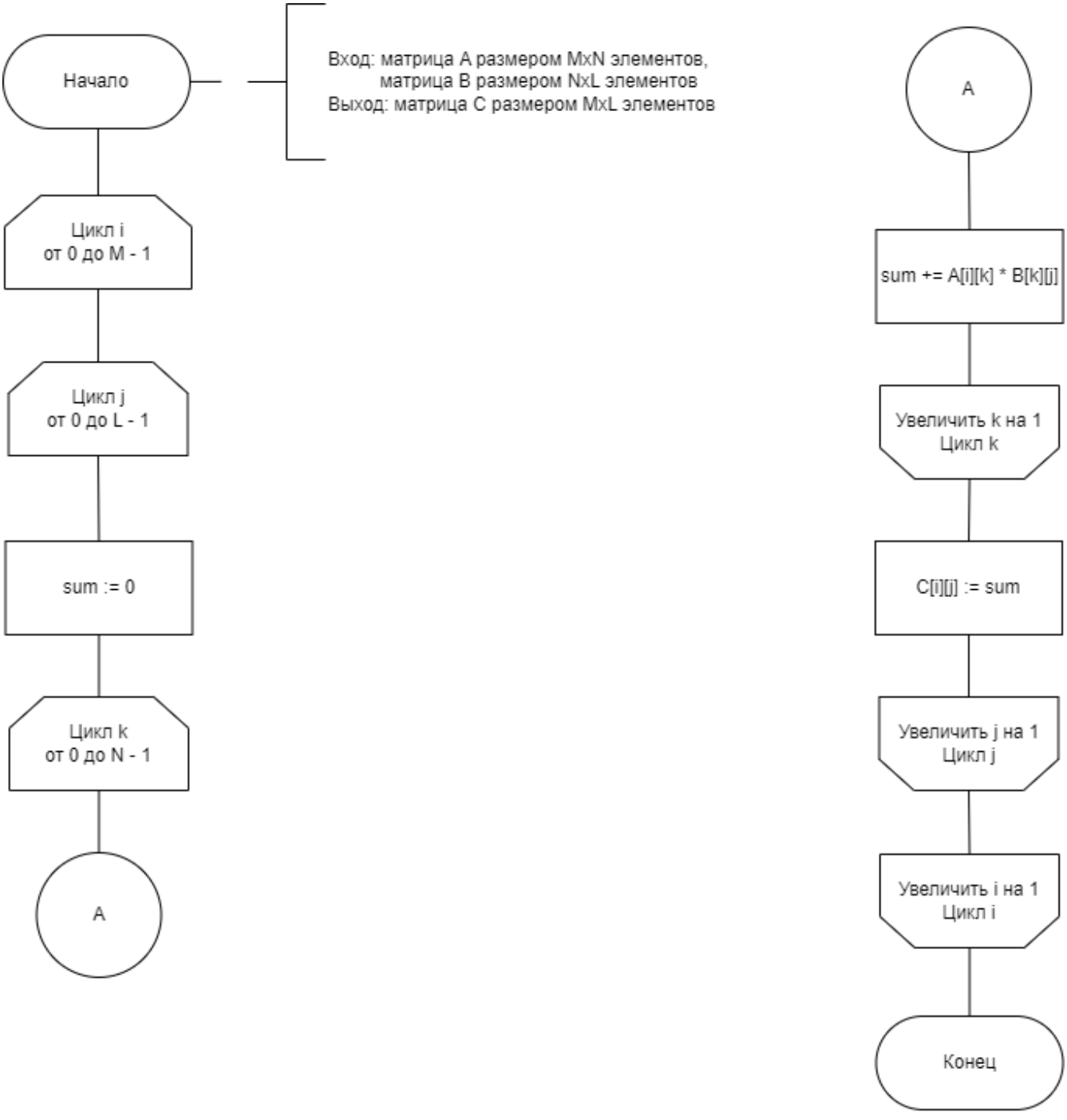

Рисунок 2.1 – Классический алгоритм умножения матриц

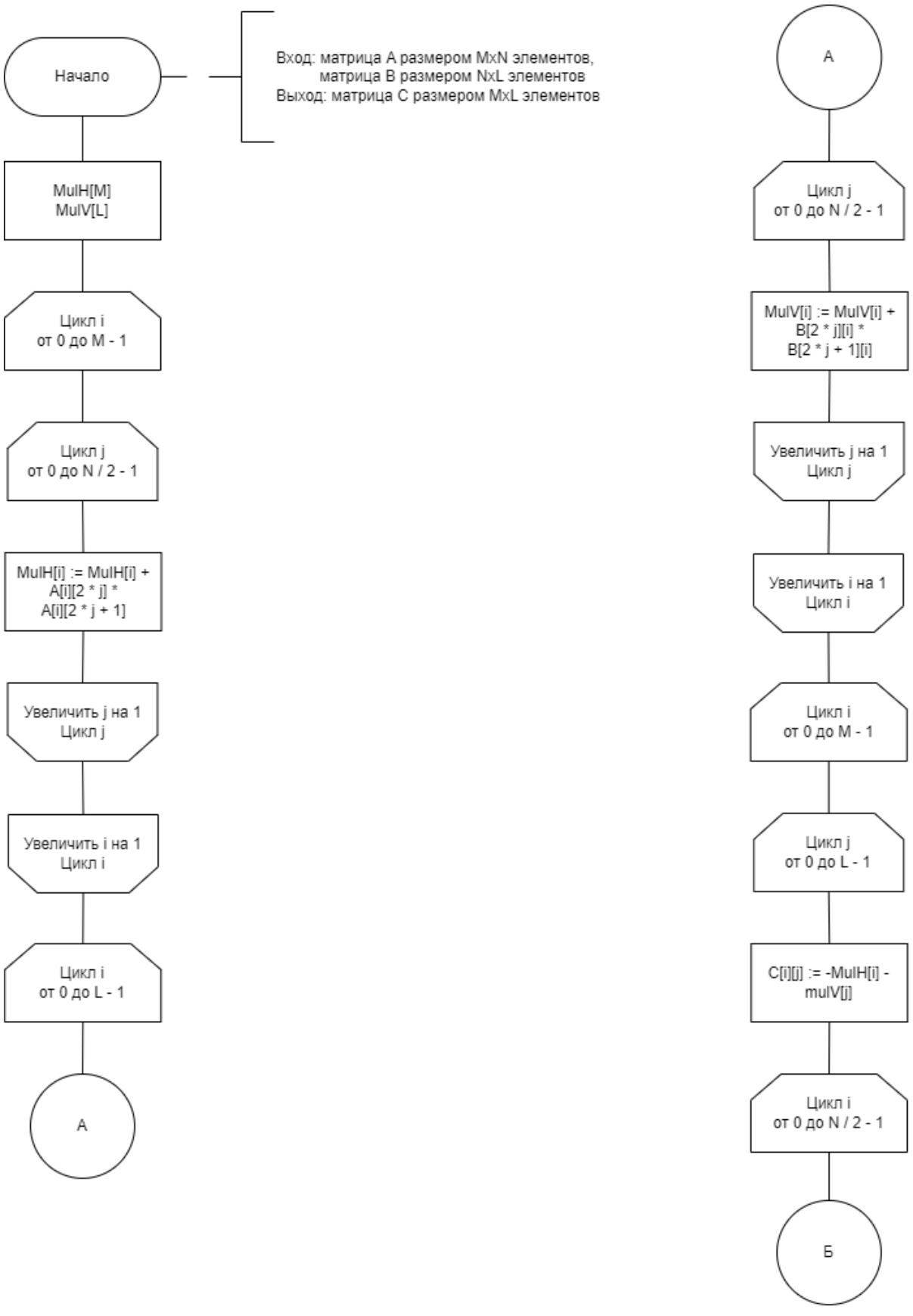

Рисунок 2.2 – Алгоритм Копперсмита–Винограда — ч. 1

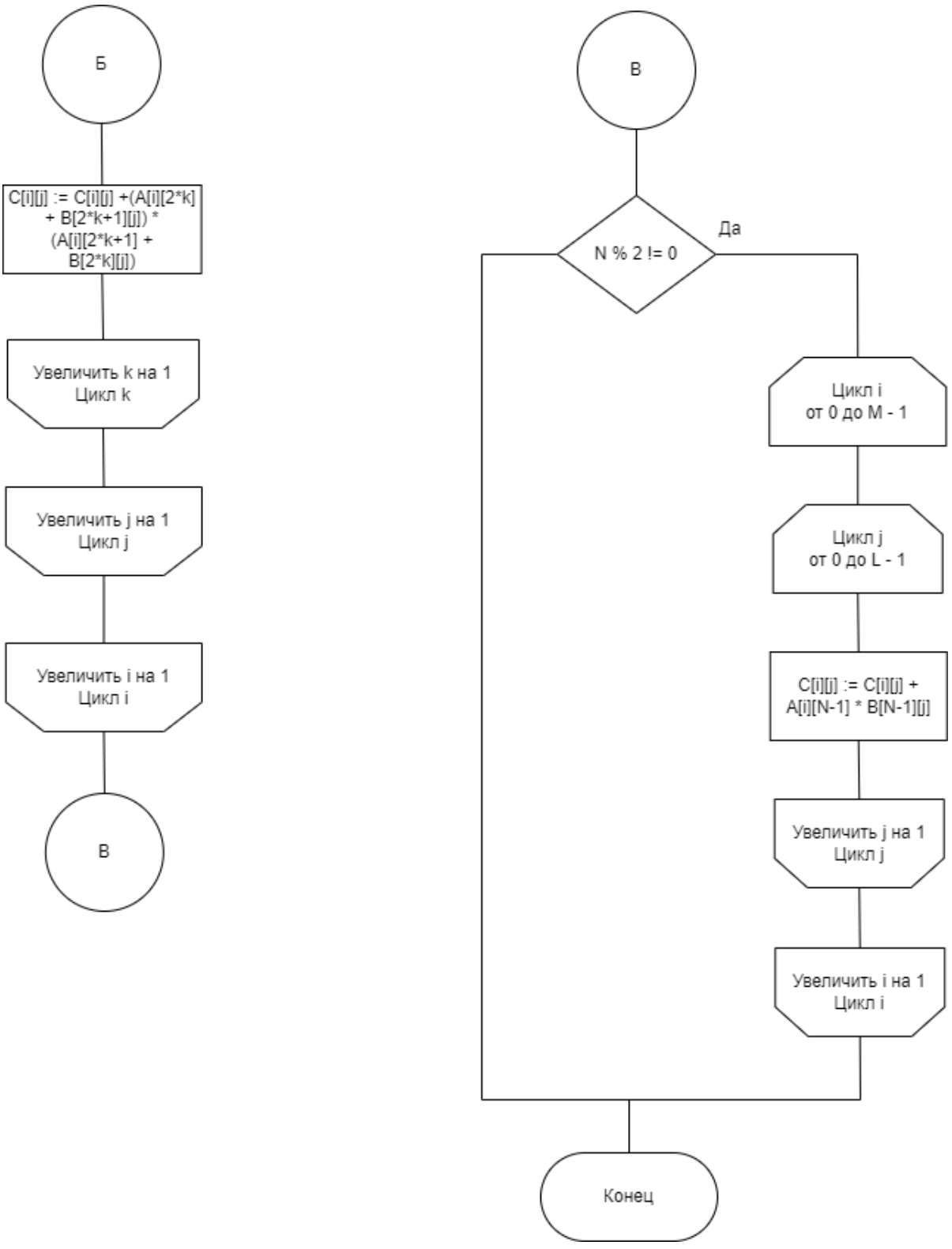

Рисунок 2.3 – Алгоритм Копперсмита–Винограда — ч. 2

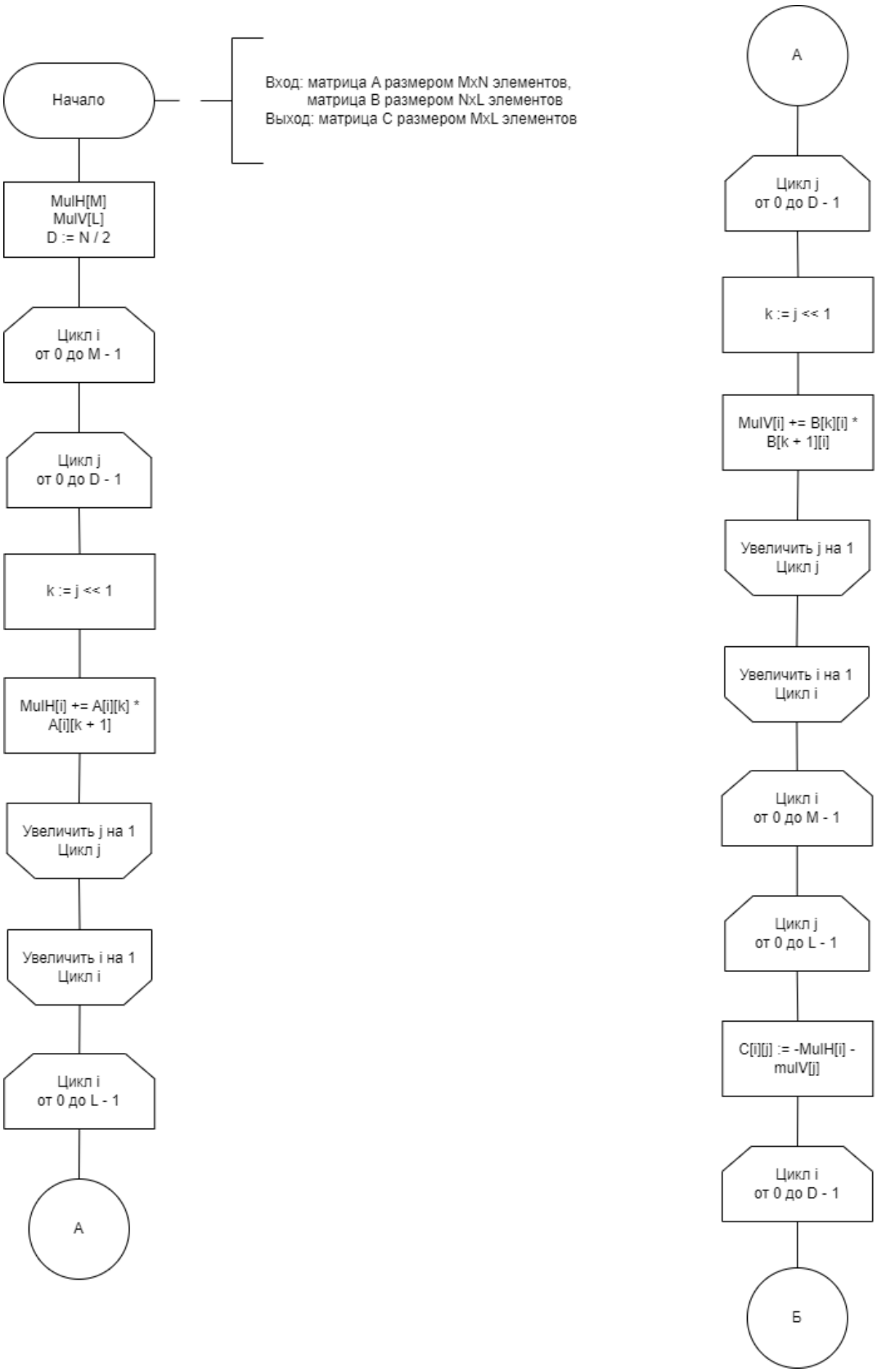

Рисунок 2.4 – Оптимизированный алгоритм Копперсмита–Винограда — ч. 1

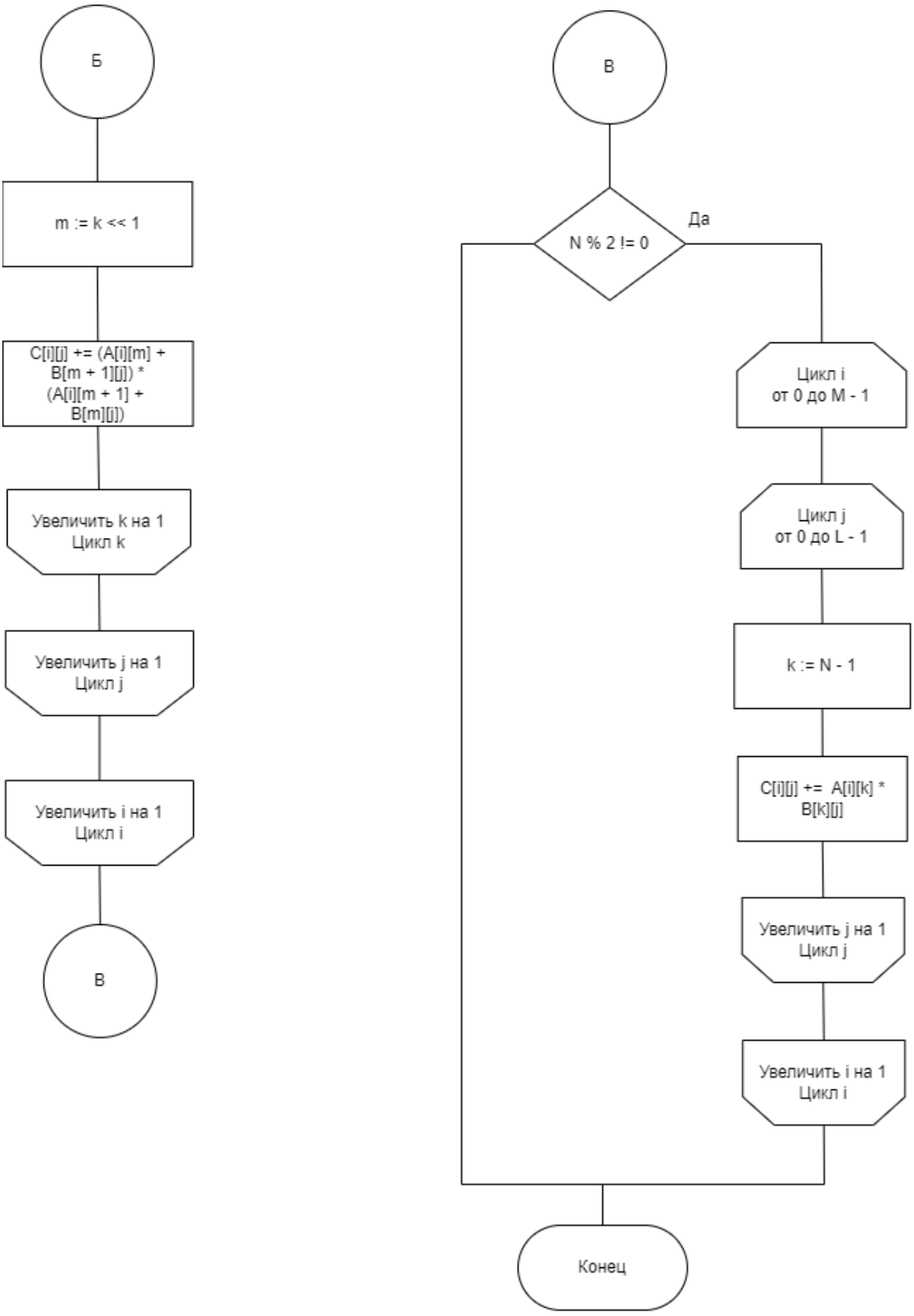

Рисунок 2.5 – Оптимизированный алгоритм Копперсмита–Винограда — ч. 2

2.2 Модель вычислений

Пусть M — количество строк в первой матрице, N — количество столбцов в первой матрице и количество строк во второй, а L — количество столбцов во второй матрице.

Для вычисления трудоемкости введена следующая модель вычислений.

1. Операции $>>, <<, [], +, -, =, +=, -=, ==, <, >, <=, >=, !=, ++, --$ имеют трудоемкость 1.

2. Операции $*, \%, /$ имеют трудоемкость 2.

3. Трудоемкость условного оператора рассчитывается следующим образом:

$$f_{if} = f_{\text{условия}} + \begin{cases} f_A, & \text{если условие выполняется,} \\ f_B, & \text{иначе.} \end{cases} \tag{2.1}$$

4. Трудоемкость цикла рассчитывается как:

$$f_{loop} = f_{\text{инициализации}} + f_{\text{сравнения}} + N \cdot (f_{\text{тела}} + f_{\text{инкремента}} + f_{\text{сравнения}}). \tag{2.2}$$

5. Трудоемкость вызова функции равна 0.

2.3 Трудоемкость алгоритмов

2.3.1 Классический алгоритм

Проверка на равенство количества столбцов в первой матрице и количества строк во второй:

$$f_1 = 1. \tag{2.3}$$

Второй вложенный цикл:

$$f_2 = 2 + 9 \cdot N. \tag{2.4}$$

Первый вложенный цикл:

$$f_3 = 2 + L \cdot (6 + f_2) = 2 + 8 \cdot L + 9 \cdot N \cdot L. \tag{2.5}$$

Внешний цикл:

$$f_4 = 2 + M \cdot (2 + f_3) = 2 + 4 \cdot M + 8 \cdot M \cdot L + 9 \cdot M \cdot N \cdot L. \tag{2.6}$$

Таким образом, трудоемкость классического алгоритма умножения матриц равна:

$$f = f_1 + f_4 = 3 + 4 \cdot M + 8 \cdot M \cdot L + 9 \cdot M \cdot N \cdot L \approx O(9 \cdot M \cdot N \cdot L). \tag{2.7}$$

Если при умножении обе входные матрицы квадратные и одинакового порядка, например, N, то трудоемкость классического алгоритма равна:

$$f \approx O(9 \cdot N^3). \tag{2.8}$$

2.3.2 Алгоритм Копперсмита–Винограда

Проверка на равенство количества столбцов в первой матрице и количества строк во второй:

$$f_1 = 1. \tag{2.9}$$

Первый цикл:

$$f_2 = 2 + M \cdot (2 + 2 + 4 + \frac{N}{2} \cdot (4 + 15)) = 2 + 8 \cdot M + \frac{19}{2} \cdot M \cdot N. \tag{2.10}$$

Второй цикл:

$$f_3 = 2 + L \cdot (2 + 2 + 4 + \frac{N}{2} \cdot (4 + 15)) = 2 + 8 \cdot L + \frac{19}{2} \cdot N \cdot L. \tag{2.11}$$

Третий цикл:

$$f_4 = 2 + M \cdot (2 + 2 + L \cdot (2 + 7 + 4 + \frac{N}{2} \cdot (4 + 26))) = \\ = 2 + 4 \cdot M + 13 \cdot M \cdot L + 15 \cdot M \cdot N \cdot L. \tag{2.12}$$

Если N нечетное:

$$f_5 = 2 + M \cdot (2 + 2 + L \cdot (2 + 14)) = 2 + 4 \cdot M + 16 \cdot M \cdot L. \tag{2.13}$$

Два цикла в теле условного оператора:

$$f_6 = 3 + \begin{cases} 2 + 4 \cdot M + 16 \cdot M \cdot L, \\ 0. \end{cases} \tag{2.14}$$

Таким образом, трудоемкость алгоритма Копперсмита–Винограда при нечетном N:

$$f = f_1 + f_2 + f_3 + f_4 + f_5 + f_6 = \\ = 12 + 16 \cdot M + 8 \cdot L + \frac{19}{2} \cdot M \cdot N + \frac{19}{2} \cdot N \cdot L + \\ + 29 \cdot M \cdot L + 15 \cdot M \cdot N \cdot L \approx O(15 \cdot M \cdot N \cdot L). \tag{2.15}$$

Трудоемкость алгоритма Копперсмита–Винограда при четном N:

$$f = f_1 + f_2 + f_3 + f_4 + f_5 + f_6 = \\ = 10 + 12 \cdot M + 8 \cdot L + \frac{19}{2} \cdot M \cdot N + \frac{19}{2} \cdot N \cdot L + \\ + 13 \cdot M \cdot L + 15 \cdot M \cdot N \cdot L \approx O(15 \cdot M \cdot N \cdot L). \tag{2.16}$$

Если при умножении обе входные матрицы квадратные и одинакового порядка, например, N, то трудоемкость равна:

$$f \approx O(15 \cdot N^3). \tag{2.17}$$

2.3.3 Оптимизированный алгоритм Копперсмита–Винограда

Проверка на равенство количества столбцов в первой матрице и количества строк во второй, а также инициализация переменной, равной $\frac{N}{2}$:

$$f_1 = 1 + 3 = 4. \tag{2.18}$$

Первый цикл:

$$f_2 = 2 + M \cdot (2 + 2 + 2 + \frac{N}{2} \cdot (2 + 11)) = 2 + 6 \cdot M + \frac{13}{2} \cdot M \cdot N. \tag{2.19}$$

Второй цикл:

$$f_3 = 2 + L \cdot (2 + 2 + 2 + \frac{N}{2} \cdot (2 + 11)) = 2 + 6 \cdot L + \frac{13}{2} \cdot N \cdot L. \tag{2.20}$$

Третий цикл:

$$\begin{gathered} f_4 = 2 + M \cdot (2 + 2 + L \cdot (2 + 7 + 2 + \frac{N}{2} \cdot (2 + 18))) = \\ = 2 + 4 \cdot M + 11 \cdot M \cdot L + 10 \cdot M \cdot N \cdot L. \end{gathered} \tag{2.21}$$

Если N нечетное:

$$f_5 = 2 + M \cdot (2 + 2 + L \cdot (2 + 12)) = 2 + 4 \cdot M + 14 \cdot M \cdot L. \tag{2.22}$$

Два цикла в теле условного оператора:

$$f_6 = 3 + \begin{cases} 2 + 4 \cdot M + 14 \cdot M \cdot L, \\ 0. \end{cases} \tag{2.23}$$

Таким образом, трудоемкость оптимизированного алгоритма Копперсмита–

Винограда при нечетном N:

$$\begin{gathered} f = f_1 + f_2 + f_3 + f_4 + f_5 + f_6 = \\ = 15 + 14 \cdot M + 6 \cdot L + \frac{13}{2} \cdot M \cdot N + \frac{13}{2} \cdot N \cdot L + \\ + 25 \cdot M \cdot L + 10 \cdot M \cdot N \cdot L \approx O(10 \cdot M \cdot N \cdot L). \end{gathered} \tag{2.24}$$

Трудоемкость оптимизированного алгоритма Копперсмита–Винограда при четном N:

$$\begin{gathered} f = f_1 + f_2 + f_3 + f_4 + f_5 + f_6 = \\ = 13 + 10 \cdot M + 6 \cdot L + \frac{13}{2} \cdot M \cdot N + \frac{13}{2} \cdot N \cdot L + \\ + 11 \cdot M \cdot L + 10 \cdot M \cdot N \cdot L \approx O(10 \cdot M \cdot N \cdot L). \end{gathered} \tag{2.25}$$

Если при умножении обе входные матрицы квадратные и одинакового порядка, например, N, то трудоемкость равна:

$$f \approx O(10 \cdot N^3). \tag{2.26}$$

Вывод из конструкторской части

В текущем разделе на основе теоретических данных, полученных из аналитического раздела, были построены схемы трех алгоритмов умножения матриц — классического, Копперсмита–Винограда и оптимизированного алгоритма Копперсмита–Винограда, а также оценены их трудоемкости.

Исходя из полученных данных, наиболее эффективным должен быть классический алгоритм умножения матриц, на втором месте — оптимизированный алгоритм Копперсмита–Винограда, а на третьем — его обычная версия.

3 Технологическая часть

В текущем разделе приведены средства реализации алгоритмов умножения матриц и листинги кода.

3.1 Требования к программному обеспечению

Программа должна запрашивать у пользователя размеры входных матриц, их элементы (целые числа), а также порядковый номер алгоритма умножения матриц. Затем на экран должен быть выведен результат умножения — новая матрица.

Программа должна обрабатывать ошибки (например, попытку ввода отрицательного количества строк в матрице) и корректно завершать работу с выводом информации об ошибке на экран.

3.2 Средства реализации

Для реализации программного обеспечения был выбран язык C++ ввиду следующих причин:

1) в библиотеке стандартных шаблонов имеется контейнер **std::vector**, который можно использовать для создания матриц;

2) есть возможность определять свои структуры данных с помощью ключевого слова **struct**;

3) для считывания данных и вывода их на экран в стандартной библиотеке существуют соответственно функции **scanf()** и **printf()**.

Таким образом, с помощью языка C++ можно реализовать программное обеспечение, которое соответствует перечисленным выше требованиям.

3.3 Реализация алгоритмов

В листинге 3.1 показана реализация матрицы.

Листинг 3.1 – Реализация матрицы

```
1 typedef struct
```

```
{
    std::vector<std::vector<int>> elements;
    std::size_t rows; // количество строк
    std::size_t columns; // количество столбцов
} matrix_t;
```

3.3.1 Классический алгоритм

В листинге 3.2 показана реализация классического алгоритма умножения матриц.

Листинг 3.2 – Реализация классического алгоритма умножения матриц

```
int default_mul(const matrix_t &matrix_1, const matrix_t
   &matrix_2, matrix_t &matrix_res)
{
    if (matrix_1.columns != matrix_2.rows)
        return 1;
    matrix_res.rows = matrix_1.rows;
    matrix_res.columns = matrix_2.columns;
    std::size_t rows = matrix_1.rows;
    std::size_t rows_and_columns = matrix_1.columns;
    std::size_t columns = matrix_2.columns;
    for (std::size_t i = 0; i < rows; i++)
    {
        std::vector<int> row;
        for (std::size_t j = 0; j < columns; j++)
        {
            int sum = 0;
            for (std::size_t k = 0; k < rows_and_columns; k++)
                sum += matrix_1.elements[i][k] *
                   matrix_2.elements[k][j];
            row.push_back(sum);
        }
        matrix_res.elements.push_back(row);
    }
    return 0;
}
```

3.3.2 Алгоритм Копперсмита–Винограда

В листинге 3.3 показана реализация алгоритма Копперсмита–Винограда.

Листинг 3.3 – Реализация алгоритма Копперсмита–Винограда

```
int winograd_mul(const matrix_t &matrix_1, const matrix_t
   &matrix_2, matrix_t &matrix_res)
{
    if (matrix_1.columns != matrix_2.rows)
        return 1;
    matrix_res.rows = matrix_1.rows;
    matrix_res.columns = matrix_2.columns;
    std::vector<int> mul_horizontal;
    std::vector<int> mul_vertical;
    std::size_t rows = matrix_1.rows;
    std::size_t rows_and_columns = matrix_1.columns;
    std::size_t columns = matrix_2.columns;
    for (std::size_t i = 0; i < rows; i++)
    {
        mul_horizontal.push_back(0);
        for (std::size_t j = 0; j < rows_and_columns / 2; j++)
            mul_horizontal[i] = mul_horizontal[i] +
                                matrix_1.elements[i][2 * j] *
                                matrix_1.elements[i][2 * j + 1];
    }
    for (std::size_t i = 0; i < columns; i++)
    {
        mul_vertical.push_back(0);
        for (std::size_t j = 0; j < rows_and_columns / 2; j++)
            mul_vertical[i] = mul_vertical[i] +
                              matrix_2.elements[2 * j][i] *
                              matrix_2.elements[2 * j + 1][i];
    }
    for (std::size_t i = 0; i < rows; i++)
    {
        std::vector<int> row;
        for (std::size_t j = 0; j < columns; j++)
        {
            row.push_back(-mul_horizontal[i] - mul_vertical[j]);
```

```
34              for (std::size_t k = 0; k < rows_and_columns / 2; k++)
35              {
36                  row[j] = row[j] + (matrix_1.elements[i][2 * k] +
37                  matrix_2.elements[2 * k + 1][j]) *
38                  (matrix_1.elements[i][2 * k + 1] +
39                  matrix_2.elements[2 * k][j]);
40              }
41          }
42          matrix_res.elements.push_back(row);
43      }
44      if (rows_and_columns % 2 != 0)
45      {
46          for (std::size_t i = 0; i < rows; i++)
47          {
48              for (std::size_t j = 0; j < columns; j++)
49                  matrix_res.elements[i][j] =
                       matrix_res.elements[i][j] +
50                  matrix_1.elements[i][rows_and_columns - 1] *
51                  matrix_2.elements[rows_and_columns - 1][j];
52          }
53      }
54      return 0;
55  }
```

3.3.3 Оптимизированный алгоритм Копперсмита–Винограда

В листинге 3.4 показана реализация оптимизированного алгоритма Копперсмита–Винограда.

Листинг 3.4 – Реализация оптимизированного алгоритма Копперсмита–Винограда

```
1  int optimized_winograd_mul(const matrix_t &matrix_1, const
       matrix_t &matrix_2, matrix_t &matrix_res)
2  {
3      if (matrix_1.columns != matrix_2.rows)
4          return 1;
5      matrix_res.rows = matrix_1.rows;
```

```
    matrix_res.columns = matrix_2.columns;
    std::vector<int> mul_horizontal;
    std::vector<int> mul_vertical;
    std::size_t rows = matrix_1.rows;
    std::size_t rows_and_columns = matrix_1.columns;
    std::size_t columns = matrix_2.columns;
    std::size_t d = rows_and_columns / 2;
    for (std::size_t i = 0; i < rows; i++)
    {
        mul_horizontal.push_back(0);
        for (std::size_t j = 0; j < d; j++)
        {
            std::size_t index = j << 1;
            mul_horizontal[i] += matrix_1.elements[i][index] *
            matrix_1.elements[i][index + 1];
        }
    }
    for (std::size_t i = 0; i < columns; i++)
    {
        mul_vertical.push_back(0);
        for (std::size_t j = 0; j < d; j++)
        {
            std::size_t index = j << 1;
            mul_vertical[i] += matrix_2.elements[index][i] *
            matrix_2.elements[index + 1][i];
        }
    }
    for (std::size_t i = 0; i < rows; i++)
    {
        std::vector<int> row;
        for (std::size_t j = 0; j < columns; j++)
        {
            row.push_back(-mul_horizontal[i] - mul_vertical[j]);
            for (std::size_t k = 0; k < d; k++)
            {
                std::size_t index = k << 1;
                row[j] += (matrix_1.elements[i][index] +
                matrix_2.elements[index + 1][j]) *
                (matrix_1.elements[i][index + 1] +
```

```
45                  matrix_2.elements[index][j]);
46              }
47          }
48          matrix_res.elements.push_back(row);
49      }
50      if (rows_and_columns % 2 != 0)
51      {
52          for (std::size_t i = 0; i < rows; i++)
53          {
54              for (std::size_t j = 0; j < columns; j++)
55              {
56                  std::size_t index = rows_and_columns - 1;
57                  matrix_res.elements[i][j] +=
58                  matrix_1.elements[i][index] *
59                  matrix_2.elements[index][j];
60              }
61          }
62      }
63      return 0;
64  }
```

3.4 Тестовые данные

В таблице 3.1 приведены тестовые данные для трех функций, реализующих алгоритмы умножения матриц.

Описание тестов:

1) пустые входные матрицы;

2) входные матрицы состоят из одного элемента;

3) квадратные входные матрицы, количество столбцов первой матрицы четное;

4) квадратные входные матрицы, количество столбцов первой матрицы нечетное;

5) неквадратные входные матрицы.

Тесты выполнялись по методологии черного ящика (модульное тестирование). Все тесты пройдены успешно.

Таблица 3.1 – Тестовые данные

Первая матрица	**Вторая матрица**	**Выходная матрица**
NULL	NULL	NULL
$\begin{bmatrix} 5 \end{bmatrix}$	$\begin{bmatrix} -9 \end{bmatrix}$	$\begin{bmatrix} -45 \end{bmatrix}$
$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix}$	$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix}$	$\begin{bmatrix} 90 & 100 & 110 & 120 \\ 202 & 228 & 254 & 280 \\ 314 & 356 & 398 & 440 \\ 426 & 484 & 542 & 600 \end{bmatrix}$
$\begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 6 & 7 & 8 & 9 & 10 \\ 11 & 12 & 13 & 14 & 15 \\ 16 & 17 & 18 & 19 & 20 \\ 21 & 22 & 23 & 24 & 25 \end{bmatrix}$	$\begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 6 & 7 & 8 & 9 & 10 \\ 11 & 12 & 13 & 14 & 15 \\ 16 & 17 & 18 & 19 & 20 \\ 21 & 22 & 23 & 24 & 25 \end{bmatrix}$	$\begin{bmatrix} 215 & 230 & 245 & 260 & 275 \\ 490 & 530 & 570 & 610 & 650 \\ 765 & 830 & 895 & 960 & 1025 \\ 1040 & 1130 & 1220 & 1310 & 1400 \\ 1315 & 1430 & 1545 & 1660 & 1775 \end{bmatrix}$
$\begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 6 & 7 & 8 & 9 & 10 \\ 11 & 12 & 13 & 14 & 15 \\ 16 & 17 & 18 & 19 & 20 \\ 21 & 22 & 23 & 24 & 25 \\ 26 & 28 & 28 & 29 & 30 \end{bmatrix}$	$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \\ 17 & 18 & 19 & 20 \end{bmatrix}$	$\begin{bmatrix} 175 & 190 & 205 & 220 \\ 400 & 440 & 480 & 520 \\ 625 & 690 & 755 & 820 \\ 850 & 940 & 1030 & 1120 \\ 1075 & 1190 & 1305 & 1420 \\ 1300 & 1440 & 1580 & 1720 \end{bmatrix}$

Вывод из технологической части

В данном разделе был написан исходный код трех алгоритмов умножения матриц — классического, Копперсмита–Винограда и оптимизированного алгоритма Копперсмита–Винограда. Описаны тесты и приведены результаты тестирования.

4 Исследовательская часть

4.1 Технические характеристики устройства

Технические характеристики устройства, на котором было проведено измерение времени работы алгоритмов:

1) операционная система Windows 10 Pro x64;
2) оперативная память 8 ГБ;
3) процессор Intel® Core™ i5-11300H @ 3.10 ГГц.

4.2 Время работы алгоритмов

Время работы функций замерены с помощью ассемблерной инструкции **rdtsc**, которая читает счетчик Time Stamp Counter и возвращает 64-битное количество тиков с момента последнего сброса процессора.

В таблице 4.1 приведено время работы в тиках трех функций, реализующих алгоритмы умножения матриц при различном количестве элементов во входных матрицах (обе входные матрицы квадратные и одинакового порядка). На рисунке 4.1 изображена зависимость времени работы в тиках алгоритмов умножения матриц от количества элементов во входных матрицах.

Таблица 4.1 – Время работы в тиках алгоритмов умножения матриц в зависимости от количества элементов во входных матрицах

Количество элементов в матрицах	**Классический алгоритм (тики)**	**Алгоритм Копперсмита–Винограда (тики)**	**Алгоритм Копперсмита–Винограда оптимизированный (тики)**
4	22808	19768	18861
9	28537	32882	31529
16	40044	48256	40338
25	63844	77181	74470
36	85947	153370	165253
49	123067	154453	168738
64	134918	179822	171832
81	150907	186049	187017
100	202561	209814	209873
121	209775	226675	215758
144	226007	263103	246680
169	237456	377059	287474
196	314170	391086	333768
225	318235	490527	437600
256	343095	661921	577530
289	524986	667464	599127
324	549017	755572	609720
361	562894	771701	658774
400	599735	850421	670344
441	711790	896018	702033

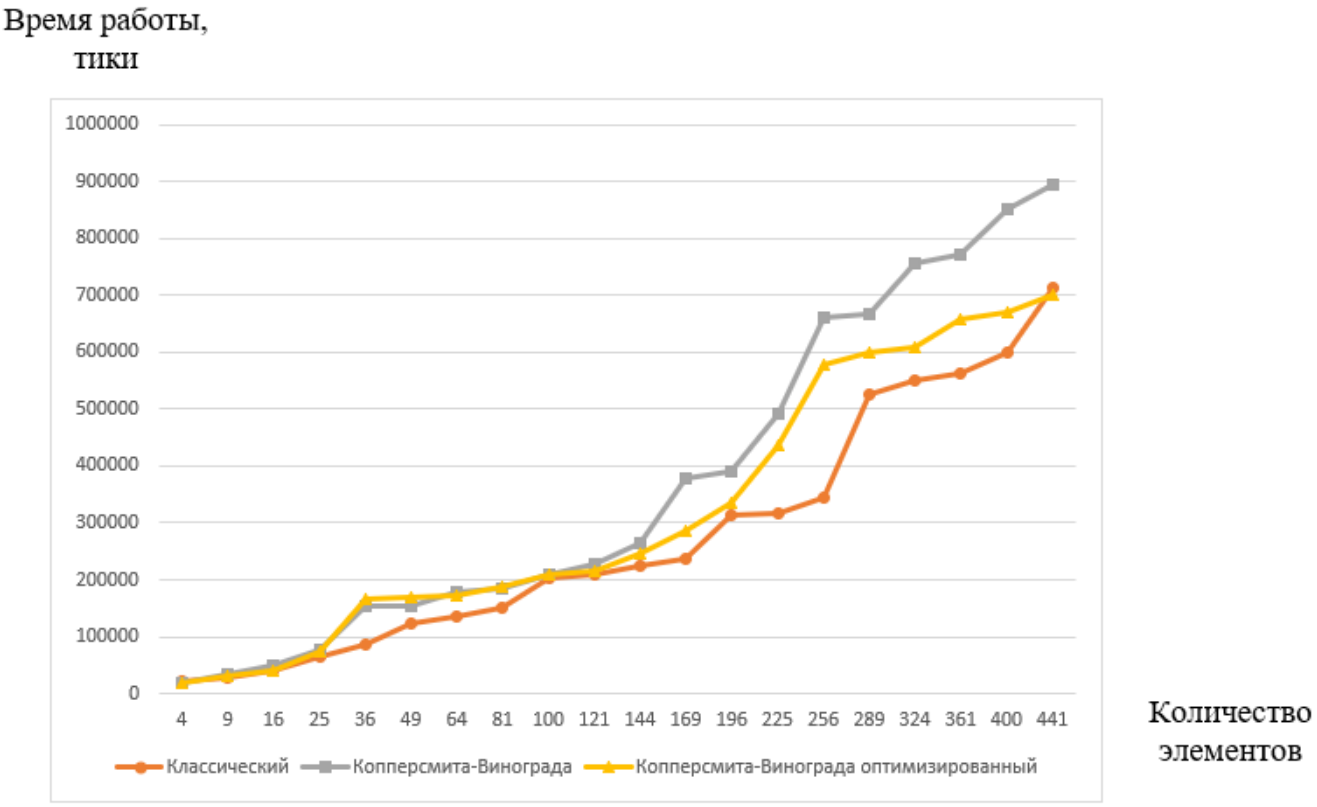

Рисунок 4.1 – Зависимость времени работы в тиках алгоритмов умножения матриц от количества элементов во входных матрицах

Вывод из исследовательской части

Согласно полученным при проведении эксперимента данным, наиболее эффективным можно считать классический алгоритм умножения матриц. На второе место по скорости работы можно поставить оптимизированный алгоритм Копперсмита–Винограда. Наименее эффективным оказался обычный алгоритм Копперсмита–Винограда.

Заключение

В рамках данной лабораторной работы была достигнута поставленная цель: были получены навыки оценки трудоемкости и временной эффективности на материале алгоритмов умножения матриц.

Решены все поставленные задачи:

1) были изучены и реализованы три алгоритма умножения матриц — классический, Копперсмита–Винограда и оптимизированный алгоритм Копперсмита–Винограда;

2) был проведен сравнительный анализ трудоемкости алгоритмов на основе теоретических расчетов и выбранной модели вычислений;

3) был проведен сравнительный анализ времени работы алгоритмов на основе экспериментальных данных.

Согласно полученным при проведении эксперимента данным, самым эффективным по времени работы является классический алгоритм умножения матриц, затем оптимизированный алгоритм Копперсмита–Винограда и обычный алгоритм Копперсмита–Винограда.

Список использованных источников

1. *Гантмахер Ф. Р.* Теория матриц // Наука. — 1966. — С. 8.

2. *Корн Г.*, *Корн Т.* Справочник по математике // Наука. — 1973. — С. 392.