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»

Студент ИУ7-52Б (Группа) ______ (Подпись, дата) Новиков А. А. (И. О. Фамилия)

Преподаватель ______ (Подпись, дата) Строганов Д. В. (И. О. Фамилия)

2024 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Поиск заданного значения в массиве или проверка существования этого значения в нем — часто встречающаяся задача при работе с массивами. Существуют различные алгоритмы для поиска заданного значения.

Цель лабораторной работы — исследование алгоритмов умножения матриц следующими методами:

- классическим методом;
- алгоритм Винограда;
- оптимизированного алгоритма Винограда.

Для достижения поставленной цели необходимо выполнить следующие задачи:

- разобрать указанные алгоритмы умножения матриц;
- выполнить оценку трудоемкости алгоритмов;
- реализовать алгоритмы умножения матриц: классический, алгоритм Винограда, оптимизированный алгоритм Винограда;
- выполнить замеры используемого процессорного времени алгоритмами в зависимости от размера входных матриц;
- описать полученные результаты в отчете.

1 Аналитический раздел

В данном разделе рассмотрены алгоритмы умножения матриц.

1.1 Описание алгоритмов

Пусть даны матрицы A с размером $N \times M$ и B с размерами $M \times K$. В результате умножения матрицы A на матрицу B получается матрица с размером $N \times K$.

1.1.1 Классический алгоритм умножения матриц

Пусть даны матрицы A размерностью $n \times m$ и матрица B размерностью $m \times k$:

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1m} \\ a_{21} & a_{22} & \dots & a_{2m} \\ \dots & \dots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nm} \end{pmatrix}, \quad B = \begin{pmatrix} b_{11} & b_{12} & \dots & b_{1k} \\ b_{21} & b_{22} & \dots & b_{2k} \\ \dots & \dots & \ddots & \vdots \\ b_{m1} & b_{m2} & \dots & b_{mk} \end{pmatrix}. \tag{1.1}$$

Тогда умножением матрицы A на матрицу B называется, где матрица C:

$$C = A \times B = \begin{pmatrix} c_{11} & c_{12} & \dots & c_{1k} \\ c_{21} & c_{22} & \dots & c_{2k} \\ \dots & \dots & \ddots & \vdots \\ c_{n1} & c_{n2} & \dots & c_{nk} \end{pmatrix}, \tag{1.2}$$

где

$$c_{ij} = \sum_{k=1}^{m} a_{ik} \cdot b_{kj} \quad (i = \overline{1,n},\ j = \overline{1,k}). \tag{1.3}$$

1.1.2 Алгоритм Винограда умножения матриц

Пусть даны матрицы A и B, имеющие размерность 4×4.

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix}, \quad B = \begin{pmatrix} b_{11} & b_{12} & b_{13} & b_{14} \\ b_{21} & b_{22} & b_{23} & b_{24} \\ b_{31} & b_{32} & b_{33} & b_{34} \\ b_{41} & b_{42} & b_{43} & b_{44} \end{pmatrix}. \tag{1.4}$$

Для получение очередного элемента c_{ij} матрицы C в классическом алгоритме умножения матрицы выполняется по формуле:

$$c_{ij} = \begin{pmatrix} a_{n1} & a_{n2} & a_{n3} & a_{n4} \end{pmatrix} \times \begin{pmatrix} b_{j1} \\ b_{j2} \\ b_{j3} \\ b_{j4} \end{pmatrix}, \tag{1.5}$$

где $a_{ni},\ i = \overline{1,4}$ - элементы n-ой строки матрицы A; $b_{jk},\ k = \overline{1,4}$ - элементы j-ого столбца матрицы B.

В алгоритме Винограда ускорение достигается за счет уменьшения числа дорогих операций умножения, которые заменяются операциями сложения [1]. Для этого производится предварительная обработка, при которой вычисляются и сохраняются значения, что позволяет заменить часть умножений на сложение, тем самым ускоряя вычисления:

$$c_{ij} = (a_{n1}+b_{j2})(a_{n2}+b_{j1})+(a_{n3}+b_{j4})(a_{n4}+b_{j3})-a_{n1}a_{n2}-a_{n3}a_{n4}-b_{j1}b_{j2}-b_{j3}b_{j4}, \tag{1.6}$$

где элементы $a_{n1}a_{n2},\ a_{n3}a_{n4},\ b_{j1}b_{j2},\ b_{j3}b_{j4}$ - значения, которые получаются в предварительной обработке.

ВЫВОД

В данном разделе были рассмотрены алгоритмы умножения матриц. Основные различия между алгоритмами - наличие предварительной обработки и количество операций умножения.

2 Конструкторский раздел

В данном разделе приведены схемы алгоритмов умножения матриц, проведены оценки трудоемкости алгоритмов.

2.1 Представление алгоритмов

На вход алгоритмов подаются две матрицы: A размером $N \times M$ и матрица B размером $M \times K$, где N, M, K - положительные числа. Алгоритмы должны возвращать матрицу C размером $N \times K$ - результат умножения матриц A и B.

Оптимизированный алгоритм Винограда включает следующие изменения:

— использование побитового сдвига вместо умножения на 2;

— использование $x+ = b$ вместо $x = x + b$;

— вынесении первой итерации из наиболее вложенного цикла.

На рисунках 2.1 — 2.5 приведены схемы следующих алгоритмов умножения матриц:

— классический алгоритм умножения (рисунок 2.1);

— классический алгоритм Винограда для умножения матриц (рисунки 2.2 — 2.3);

— оптимизированный алгоритм Винограда для умножения матриц (рисунки 2.4 — 2.5).

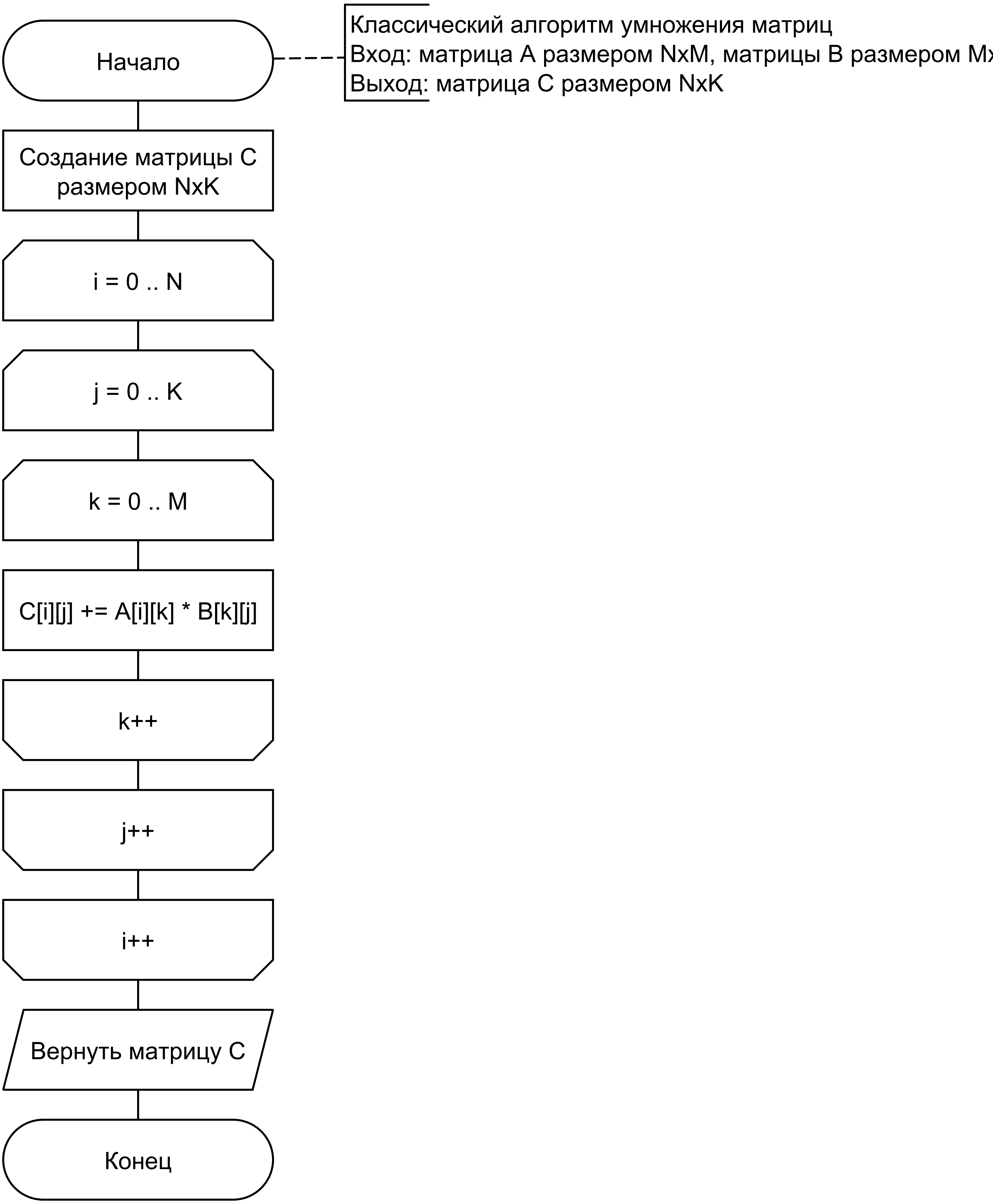

Рисунок 2.1 – Классический алгоритм умножения

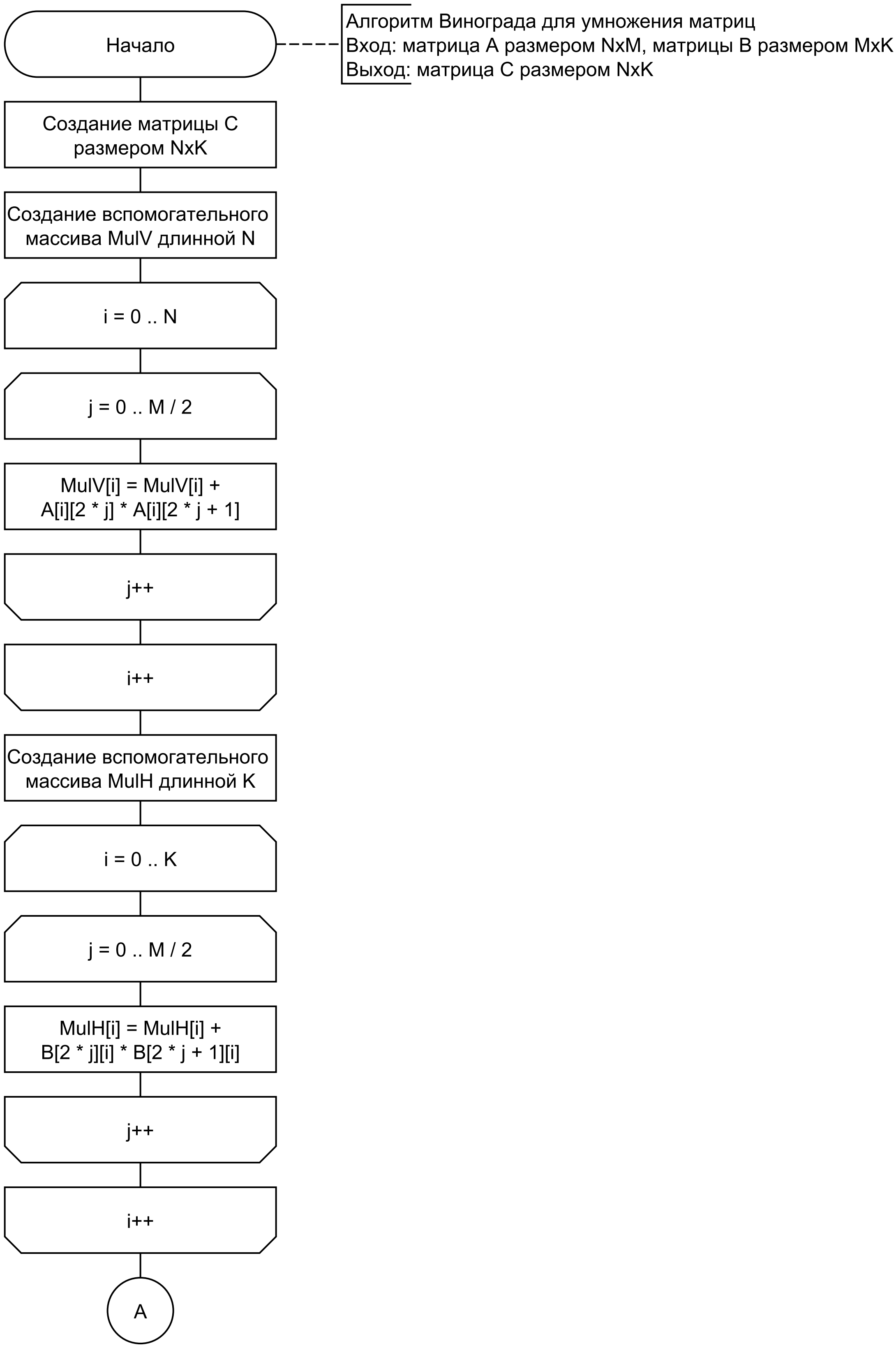

Рисунок 2.2 – Алгоритм Винограда, часть 1

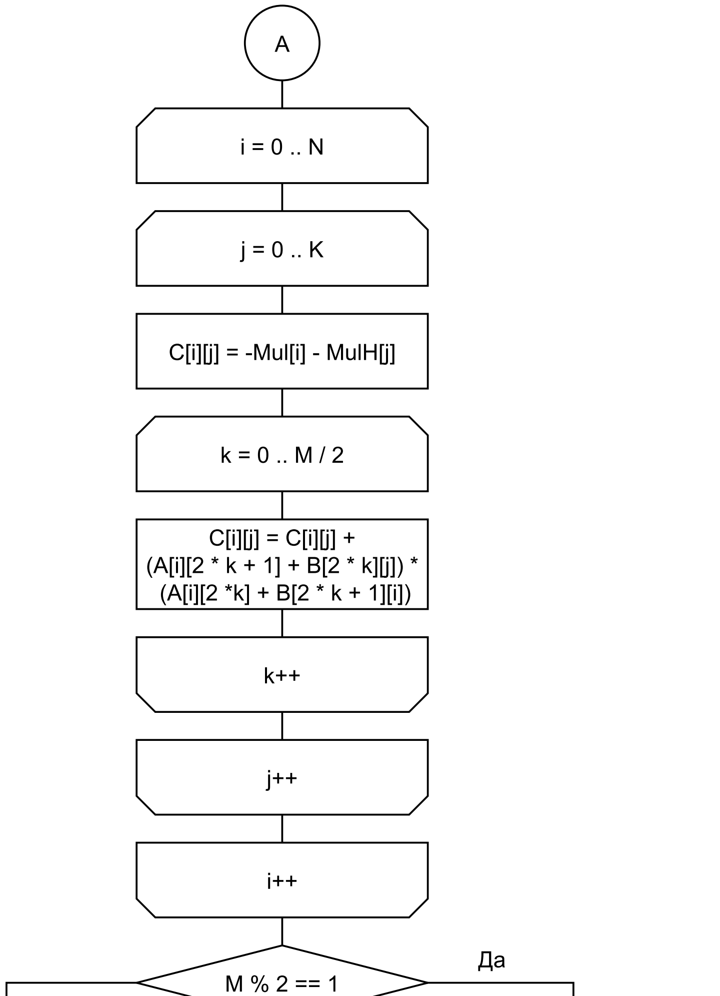

Рисунок 2.3 – Алгоритм Винограда, часть 2

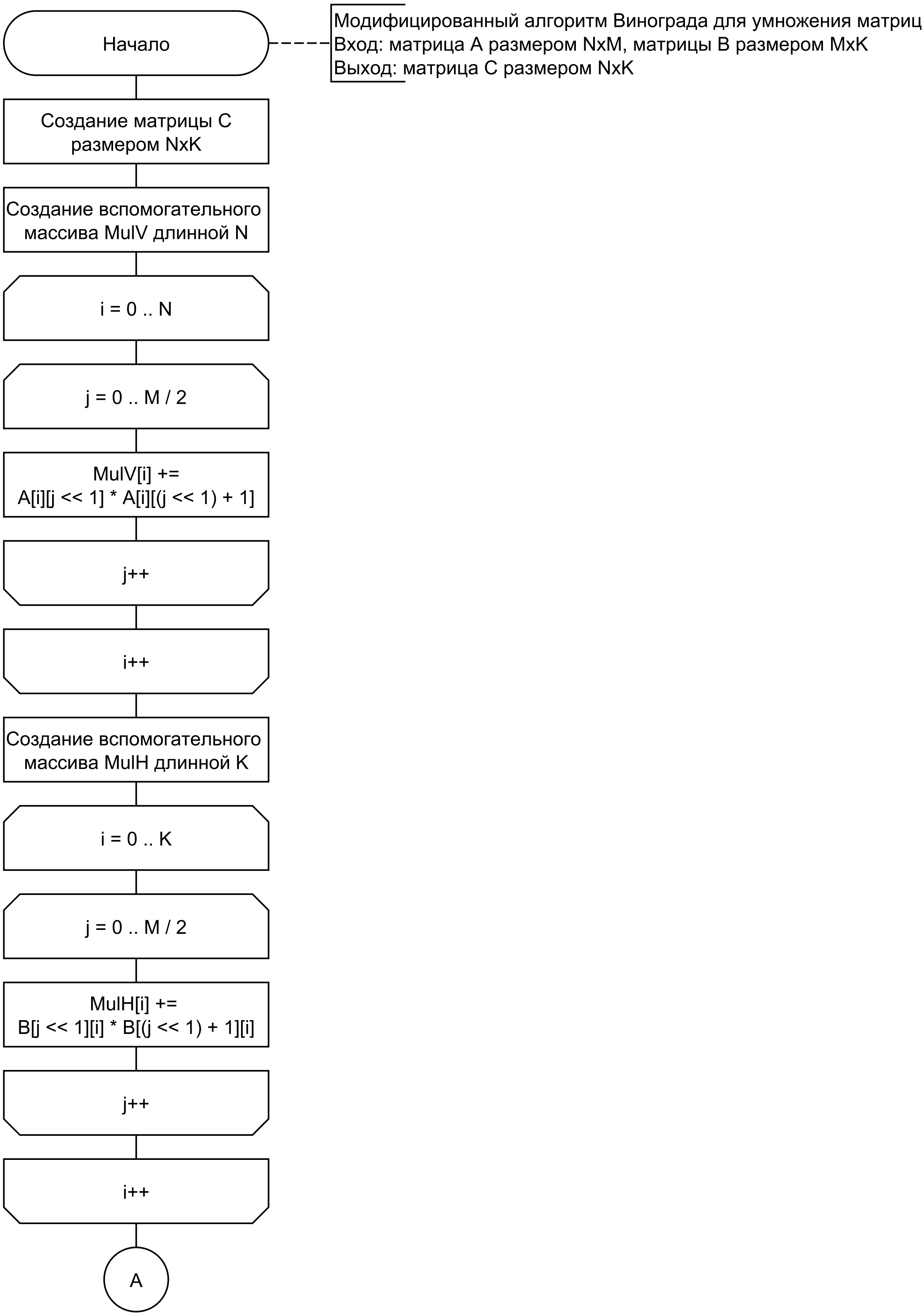

Рисунок 2.4 – Модифицированный алгоритм Винограда, часть 1

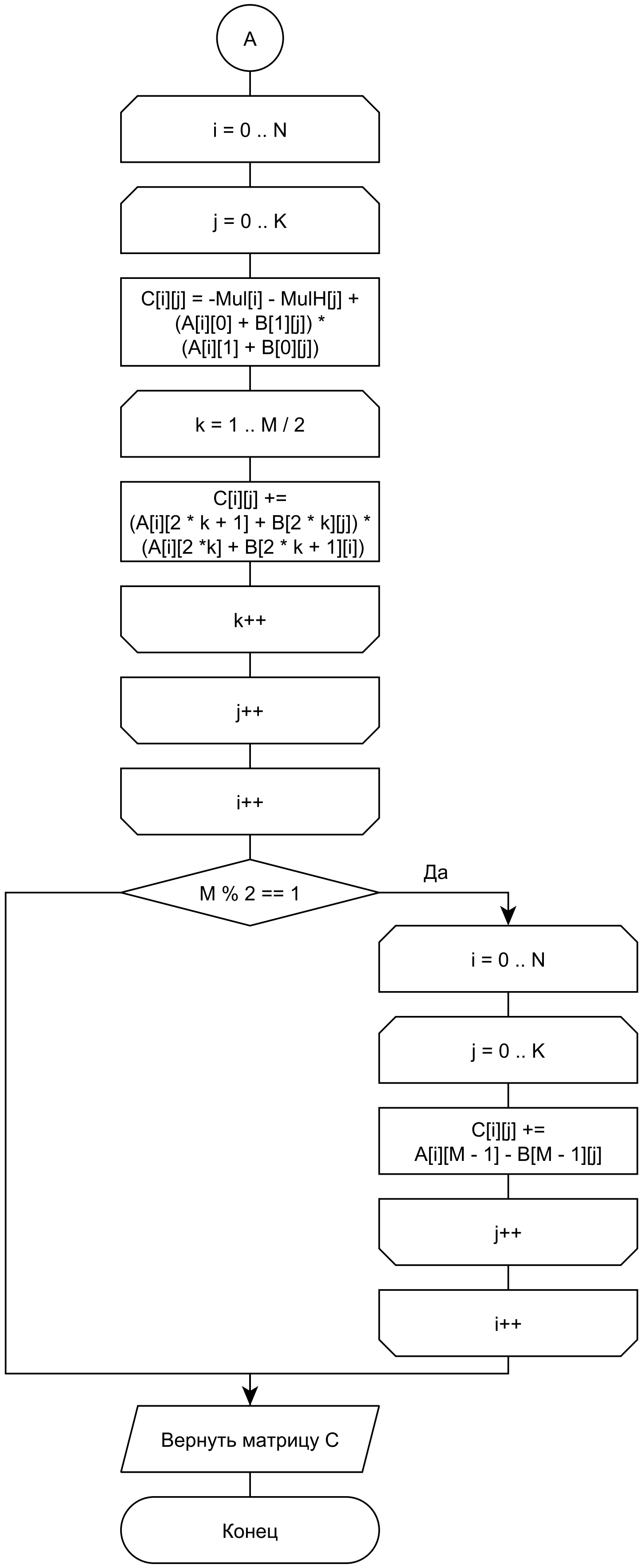

Рисунок 2.5 – Модифицированный алгоритм Винограда, часть 2

2.2 Модель вычислений

Для вычисления трудоемкости введем 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 условного оператора 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/возврата результата равна 0.

2.3 Трудоемкость алгоритмов

В следующих частях будут рассчитаны трудоемкости представленных ранее классического алгоритма, алгоритма Винограда, оптимизированного алгоритма Винограда. Трудоемкость инициализации результирующей матрицы учитываться не будет, поскольку данное действие есть во всех алгоритмах и не является самым трудоемким.

Введем обозначения:

— N — количество строк первой матрицы;

— M — количество столбцов первой матрицы и количество строк второй матрицы;

— K — количество столбцов второй матрицы.

2.3.1 Классический алгоритм умножения матриц

Трудоемкость стандартного алгоритма умножения матриц состоит из:

— внешнего цикла по $i \in [1..N]$, трудоемкость которого: $f = 2 + N \cdot (2 + f_{body})$;

— цикла по $j \in [1..K]$, трудоемкость которого: $f = 2 + K \cdot (2 + f_{body})$;

— цикла по $k \in [1..M]$, трудоемкость которого: $f = 2 + 12 \cdot M$.

Трудоемкость классического алгоритма равна трудоемкости внешнего цикла. Ее можно вычислить, подставив циклы тела (2.5):

$$f_{classic} = 2 + N \cdot (4 + K \cdot (4 + 11M)) = 2 + 4N + 4NK + 14NMK \approx 14NMK \quad (2.5)$$

2.3.2 Алгоритм Винограда

Трудоемкость алгоритма Винограда состоит из:

— создания и инициализации массивов MulV и MulH, трудоемкость которого (2.12):

$$f_{init} = N + K; \quad (2.6)$$

— заполнения массива MulV, трудоемкость которого (2.13):

$$f_{MulV} = 2 + N \cdot (4 + \frac{M}{2} \cdot 19); \quad (2.7)$$

— заполнения массива MulH, трудоемкость которого (2.14):

$$f_{MulH} = 2 + K \cdot (4 + \frac{M}{2} \cdot 19); \quad (2.8)$$

— цикла заполнения для четных размеров, трудоемкость которого (2.15):

$$f_{cycle} = 2 + N \cdot (2 + K \cdot (2 + 7 + 4 + \frac{M}{2} \cdot (4 + 26))); \quad (2.9)$$

— цикла, для дополнения результирующего массива суммой последних нечетных строки и столбца, если общий размер нечетный, трудоемкость которого (2.16):

$$f_{last} = \begin{cases} 2, & \text{размер четный,} \\ 2 + N \cdot (2 + 14K), & \text{иначе.} \end{cases} \tag{2.10}$$

Итого, результирующая трудоемкость алгоритма Винограда равна (2.17)

$$f_{final} = f_{init} + f_{MulV} + f_{MulH} + f_{cycle} + f_{last} \approx 15NMK \tag{2.11}$$

Классический алгоритм Винограда имеет большую трудоемкость, чем классический алгоритм умножения матриц.

2.3.3 Оптимизированный алгоритм Винограда

Трудоемкость оптимизированного алгоритма Винограда состоит из:

— создания и инициализации массивов MulV и MulH а также доп. переменной, хранящей N/2, трудоемкость которого (2.12):

$$f_{init} = N + K; \tag{2.12}$$

— заполнения массива MulV, трудоемкость которого (2.13):

$$f_{MulV} = 2 + N \cdot (4 + \frac{M}{2} \cdot 11); \tag{2.13}$$

— заполнения массива MulH, трудоемкость которого (2.14):

$$f_{MulH} = 2 + K \cdot (4 + \frac{M}{2} \cdot 11); \tag{2.14}$$

— цикла заполнения для четных размеров, трудоемкость которого (2.15):

$$f_{cycle} = 2 + N \cdot (2 + K \cdot (2 + 20 + 4 + (\frac{M}{2} - 1) \cdot (4 + 20))); \tag{2.15}$$

— цикла, для дополнения результирующего массива суммой последних нечетных строки и столбца, если общий размер нечетный, трудоемкость которого (2.16):

$$f_{last} = \begin{cases} 2, & \text{размер четный,} \\ 2 + N \cdot (2 + 11K), & \text{иначе.} \end{cases} \tag{2.16}$$

Итого, результирующая трудоемкость оптимизированного алгоритма Винограда равна (2.17)

$$f_{final} = f_{init} + f_{A1} + f_{B1} + f_{cycle} + f_{last} \approx 12NMK \tag{2.17}$$

Оптимизированный алгоритм Винограда имеет меньшую трудоемкость, по сравнению с классическим алгоритмом.

ВЫВОД

В данном разделе были приведены схемы алгоритмов умножения матриц, а также были проведены расчеты трудоемкости каждого из трех алгоритмов.

3 Технологический раздел

В данном разделе описаны требования к программному обеспечению, реализация алгоритмов и средства реализации.

3.1 Требования к программному обеспечению

Входные данные: две матрицы, количество столбцов первой матрицы равно количеству строк второй матрицы; Выходные данные: матрицы, являющаяся произведением вхожных матриц.

3.2 Средства реализации

Для реализации данной лабораторной работы выбран язык программирования *C* [2]. Выбор был обусловлен необходимостью производить замеры на микроконтроллерах и наличием библиотеки *time* [3]. Время было замерено с помощью функции *clock* [4].

3.3 Реализация алгоритмов

В листингах 3.1 — 3.3 представлены реализации алгоритмов.

Листинг 3.1 – Реализация классического алгоритма

```
error_t matrix_mul(const matrix_t *matrix_1, const matrix_t *matrix_2, matrix_t *result) {
    if (matrix_1->collumn != matrix_2->row) return ERR_RANGE;
    error_t rc = alloc_matrix_t(result, matrix_1->row, matrix_2->collumn);
    if (rc) return rc;
    for (size_t i = 0; i < result->row; ++i)
    {
        for (size_t j = 0; j != result->collumn; ++j)
        {
            result->data[i][j] = 0;
            for (size_t k = 0; k < matrix_1->collumn; ++k)
                result->data[i][j] = result->data[i][j] + matrix_1->data[i][k] * matrix_2->data[k][j];
        }
    }
    return ERR_OK;
}
```

Листинг 3.2 – Реализация алгоритма Винограда

```
error_t vinograd_matrix_mul(const matrix_t *matrix_1, const
   matrix_t *matrix_2, matrix_t *result) {
    if (matrix_1->collumn != matrix_2->row)
        return ERR_RANGE;

    error_t rc = alloc_matrix_t(result, matrix_1->row,
       matrix_2->collumn);
    if (rc)
        return rc;

    size_t row_1 = matrix_1->row;
    size_t collumn_1 = matrix_1->collumn;
    size_t collumn_2 = matrix_2->collumn;

    int *row_factor = malloc(row_1 * sizeof(int));
    int *col_factor;
    if (row_factor != NULL) {
        col_factor = malloc(collumn_2 * sizeof(int));
        if (col_factor == NULL) {
            free(row_factor);
            return ERR_MEM;
        }
    } else
        return ERR_MEM;

    for (size_t i = 0; i < row_1; ++i) {
        row_factor[i] = 0;
        for (size_t j = 0; j < collumn_1 / 2; ++j) {
            row_factor[i] = row_factor[i] +
                            matrix_1->data[i][2 * j] *
                            matrix_1->data[i][2 * j + 1];
        }
    }

    for (size_t j = 0; j < collumn_2; ++j) {
        col_factor[j] = 0;
        for (size_t i = 0; i < collumn_1 / 2; ++i) {
            col_factor[j] = col_factor[j] +
                            matrix_2->data[2 * i][j] *
                            matrix_2->data[2 * i + 1][j];
```

```
        }
    }

    for (size_t i = 0; i < row_1; ++i) {
        for (size_t j = 0; j < collumn_2; ++j) {
            result->data[i][j] = -row_factor[i] - col_factor[j];
            for (size_t k = 0; k < collumn_1 / 2; ++k) {
                result->data[i][j] = result->data[i][j] +
                (matrix_1->data[i][2 * k] +
                matrix_2->data[2 * k + 1][j]) *
                (matrix_1->data[i][2 * k + 1] +
                matrix_2->data[2 * k][j]);
            }
        }
    }

    if (collumn_1 % 2 == 1) {
        for (size_t i = 0; i < row_1; ++i) {
            for (size_t j = 0; j < collumn_2; ++j) {
                result->data[i][j] = result->data[i][j] +
                matrix_1->data[i][collumn_1 - 1] *
                matrix_2->data[collumn_1 - 1][j];
            }
        }
    }

    free(col_factor);
    free(row_factor);

    return ERR_OK;
}
```

Листинг 3.3 – Реализация оптимизированного алгоритма Винограда

```
error_t vinograd_modify_matrix_mul(const matrix_t *matrix_1,
   const matrix_t *matrix_2, matrix_t *result) {
    if (matrix_1->collumn != matrix_2->row)
        return ERR_RANGE;

    error_t rc = alloc_matrix_t(result, matrix_1->row,
       matrix_2->collumn);
    if (rc)
        return rc;

    size_t row_1 = matrix_1->row;
    size_t collumn_1 = matrix_1->collumn;
    size_t collumn_2 = matrix_2->collumn;

    int *row_factor = malloc(row_1 * sizeof(int));
    int *col_factor;
    if (row_factor != NULL) {
        col_factor = malloc(collumn_2 * sizeof(int));
        if (col_factor == NULL) {
            free(row_factor);
            return ERR_MEM;
        }
    } else
        return ERR_MEM;

    for (size_t i = 0; i < row_1; ++i) {
        row_factor[i] = 0;
        for (size_t j = 0; j < collumn_1 / 2; ++j) {
            row_factor[i] += matrix_1->data[i][j << 1] *
               matrix_1->data[i][(j << 1) + 1];
        }
    }

    for (size_t j = 0; j < collumn_2; ++j) {
        col_factor[j] = 0;
        for (size_t i = 0; i < collumn_1 / 2; ++i) {
            col_factor[j] += matrix_2->data[i << 1][j] *
               matrix_2->data[(i << 1) + 1][j];
        }
    }
```

```

    for (size_t i = 0; i < row_1; ++i) {
        for (size_t j = 0; j < collumn_2; ++j) {
            result->data[i][j] = -row_factor[i] - col_factor[j] +
                                 (matrix_1->data[i][0] +
                                 matrix_2->data[1][j]) *
                                 (matrix_1->data[i][1] +
                                 matrix_2->data[0][j]);
            for (size_t k = 1; k < collumn_1 / 2; ++k) {
                result->data[i][j] +=
                (matrix_1->data[i][k << 1] +
                matrix_2->data[(k << 1) + 1][j]) *
                (matrix_1->data[i][(k << 1) + 1] +
                matrix_2->data[k << 1][j]);
            }
        }
    }

    if (collumn_1 % 2 == 1) {
        for (size_t i = 0; i < row_1; ++i) {
            for (size_t j = 0; j < collumn_2; ++j) {
                result->data[i][j] +=
                matrix_1->data[i][collumn_1 - 1] *
                matrix_2->data[collumn_1 - 1][j];
            }
        }
    }

    free(col_factor);
    free(row_factor);

    return ERR_OK;
}
```

ВЫВОД

В данном разделе были представлены реализации алгоритмов умножения матриц, были рассмотрены средства реализации, предъявлены требования к программному обеспечению.

4 Исследовательский раздел

В данном разделе проведен сравнительный анализ алгоритмов по используемому процессорному времени.

4.1 Технические характеристики

Технические характеристики используемого устройства:

— Операционная система — Windows 10 Home [5];

— Память — 16 Гб;

— Процессор — Intel(R) Core(TM) i5-10300H CPU @ 2.50 ГГц [6];

— Микроконтроллер — STM32F303 [7].

4.2 Время выполнения алгоритмов

Время работы трех алгоритмов умножения матриц было измерено и представлено в таблице 4.1. Тестирование проводилось на микроконтроллере STM32F303 с тактовой частотой до 72 МГц. Измерения выполнялись на матрицах одинакового размера и усреднялись для каждого набора однотипных экспериментов. Каждое значение является средним результатом 100 замеров. График зависимости времени умножения от размера матриц для трех алгоритмов показан на рисунке 4.1.

Таблица 4.1 – Время работы алгоритмов (в мс)

Размер матрицы	Классический	Виноград	Виноград (оптимизированный)
3	0.3	0.3	0.5
5	1.1	1.5	1.3
7	2.9	3.3	2.8
9	5.7	6.4	5.7
11	10.2	11.0	9.7
13	18.0	18.3	16.3
15	28.9	26.5	24.0
17	39.6	38.6	36.6
19	51.8	51.4	46.0
21	68.7	72.7	61.9
23	97.5	96.0	83.0
25	122.3	114.1	108.0
27	149.9	153.2	138.1
29	180.1	175.3	161.8
31	239.1	230.1	208.0
33	282.9	282.9	247.4
35	329.5	329.5	298.6

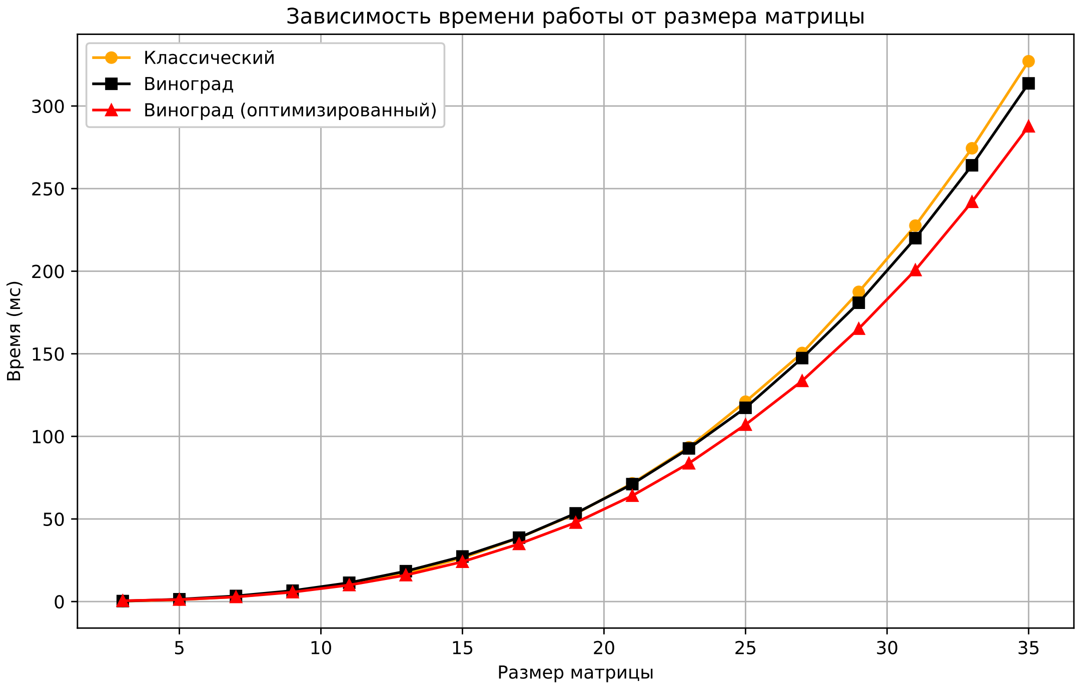

Рисунок 4.1 – Сравнение алгоритмов по времени

ВЫВОД

Исследование показало, что классический алгоритм умножения матриц уступает алгоритму Винограда по скорости примерно в 1.2 раза, поскольку в алгоритме Винограда часть вычислений выполняется заранее, а количество сложных операций, таких как умножение, уменьшается. Следовательно, алгоритм Винограда предпочтительнее. Оптимизированный алгоритм Винограда, в свою очередь, демонстрирует еще лучшее время работы — он быстрее на 1.2 раза на матрицах размером более 10 элементов благодаря замене операций сложения и присваивания, сдвигу вместо умножения и предварительным вычислениям некоторых выражений. Таким образом, для наилучшей производительности стоит выбирать оптимизированный алгоритм Винограда.

ЗАКЛЮЧЕНИЕ

В результате исследования было определено, что классический алгоритм умножения матриц проигрывает по времени алгоритму Винограда примерно в 1.2 раза из-за того, что в алгоритме Винограда часть вычислений происходит заранее, а также сокращается часть сложных операций - операций умножения, поэтому предпочтение следует отдавать алгоритму Винограда. Но лучшие показатели по времени выдает оптимизированный алгоритм Винограда – он примерно в 1.2 раза быстрее алгоритма Винограда на размерах матриц свыше 10 из-за замены операций равно и плюс на операцию плюс-равно, за счет замены операции умножения операцией сдвига, а также за счет предвычислений некоторых слагаемых, что дает проводить часть вычислений быстрее. Поэтому при выборе самого быстрого алгоритма предпочтение стоит отдавать оптимизированному алгоритму Винограда.

В ходе выполнения данной лабораторной работы были решены следующие задачи:

— разобраны алгоритмы умножения матриц: стандартный, Винограда и оптимизированный алгоритм Винограда;

— выполнена оценка трудоемкости алгоритмов;

— реализованы алгоритмы умножения матриц: стандартный, алгоритм Винограда и оптимизированный алгоритм Винограда;

— выполнены замеры используемого процессорного времени алгоритмами в зависимости от размера входных матриц;

— описаны полученные результаты в отчете.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Understanding 'Winograd Fast Convolution' [Электронный ресурс]. — URL: https://medium.com/@dmangla3/understanding-winograd-fast-convolution-a75458744ff (дата обращения: 05.10.2024).
2. C Programming Language [Электронный ресурс]. — URL: https://devdocs.io/c/ (дата обращения: 05.10.2024).
3. C Date and Time Utilities [Электронный ресурс]. — URL: https://en.cppreference.com/w/c/chrono (дата обращения 05.10.2024).
4. C clock() Documentation [Электронный ресурс]. — URL: https://en.cppreference.com/w/c/chrono/clock (дата обращения 05.10.2024).
5. Windows 10: A Guide for Advanced Users [Электронный ресурс]. — URL: https://docs.microsoft.com/en-us/windows/security/information-protection/advanced-windows-10-security (дата обращения: 05.10.2024).
6. Intel® Core™ i5-10300H Processor [Электронный ресурс]. — URL: https://ark.intel.com/content/www/us/en/ark/products/201839/intel-core-i5-10300h-processor-8m-cache-up-to-4-50-ghz.html (дата обращения 05.10.2024).
7. STM32F303 PDF Documentation [Электронный ресурс]. — URL: https://www.st.com/en/microcontrollers-microprocessors/stm32f303/documentation.html (дата обращения 05.10.2024).