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»

НАПРАВЛЕНИЕ ПОДГОТОВКИ «09.03.04 Программная инженерия»

ОТЧЕТ

ПО ЛАБОРАТОРНОЙ РАБОТЕ №2

Тема: Умножение матриц

Студент: Сироткина П.Ю.

Группа: ИУ7-56Б

Оценка: ______________

Преподаватель: Волкова Л.Л.

Москва, 2021 г.

Содержание

Введение 3
1 Аналитический раздел 5
1.1 Стандартный алгоритм 5
1.2 Алгоритм Копперсмита-Винограда 6
1.3 Вывод 7
2 Конструкторский раздел 8
2.1 Схемы алгоритмов 8
2.2 Трудоемкость алгоритмов 15
2.3 Вывод 18
3 Технологический раздел 19
3.1 Требования к ПО 19
3.2 Средства реализации 19
3.3 Листинг кода 19
3.4 Вывод 27
4 Экспериментальный раздел 28
4.1 Пример работы 28
4.2 Тестовые данные 29
4.3 Технические характеристики 29
4.4 Время выполнения алгоритмов 30
4.5 Вывод 33
Заключение 34
Список использованных источников 35

Введение

Матричное умножение — это один из базовых алгоритмов, который широко применяется в различных численных методах, в частности в алгоритмах машинного обучения.

Матричное умножение позволяет эффективно задействовать все вычислительные ресурсы современных процессоров и графических ускорителей, поэтому не удивительно, что многие алгоритмы стараются свести к матричному умножению — дополнительная расходы, связанные с подготовкой данных, как правило с лихвой окупаются общим ускорением алгоритмов.

В данной лабораторной работе будут изучены и реализованы некоторые методы умножения матриц, а именно: классический алгоритм, соответствующей математической формулировке, и алгоритм Копперсмита-Винограда, а также их оптимизированные версии.

Алгоритм Копперсмита—Винограда — алгоритм умножения квадратных матриц, предложенный в 1987 году Д. Копперсмитом и Ш. Виноградом (англ.). В исходной версии асимптотическая сложность алгоритма составляла $O(n^{2.3755})$, где n — размер стороны матрицы. Алгоритм Копперсмита—Винограда, с учетом серии улучшений и доработок в последующие годы, обладает лучшей асимптотикой среди известных алгоритмов умножения матриц.[1]

На практике алгоритм Копперсмита—Винограда не используется, так как он имеет очень большую константу пропорциональности и начинает выигрывать в быстродействии у других известных алгоритмов только для матриц, размер которых превышает память современных компьютеров.

Целью лабораторной работы является изучение и реализация алгоритмов умножения матриц.

Для достижения поставленной цели необходимо выполнить следующие **задачи**:

а) Изучить и реализавать методы умножения матриц: стандартный метод и метод Винограда, а также его оптимизированную версию.

б) Рассчитать трудоемкость реализованных алгоритмов.

в) Провести сравнительный анализ затрачиваемого процессорного времени реализованных алгоритмов.

г) На основании выполненной работы сделать выводы.

1 Аналитический раздел

Матрица — это математический объект, записываемый в виде прямоугольной таблицы элементов кольца или поля (например, целых, действительных или комплексных чисел), который представляет собой совокупность строк и столбцов, на пересечении которых находятся его элементы. Количество строк и столбцов задает размер матрицы.

Умножение матриц — это один из базовых алгоритмов, который широко применяется в различных численных методах, в частности в алгоритмах машинного обучения. На умножение матриц наложено условие: матрицу, имеющую n столбцов, можно умножить справа на матрицу, имеющую n строк. Важно отметить, что умножение матриц *некоммутативно*.

1.1 Стандартный алгоритм

Пусть даны две прямоугольные матрицы:

$$A_{mn} = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{l2} & \dots & a_{mn} \end{pmatrix}, \quad B_{nk} = \begin{pmatrix} b_{11} & b_{12} & \dots & b_{1k} \\ b_{21} & b_{22} & \dots & b_{2k} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \dots & b_{nk} \end{pmatrix}, \tag{1.1}$$

Тогда матрица C, определенная формулой (1.2), будет называться произведением матриц A и B:

$$C_{mk} = \begin{pmatrix} c_{11} & c_{12} & \dots & c_{1k} \\ c_{21} & c_{22} & \dots & c_{2k} \\ \vdots & \vdots & \ddots & \vdots \\ c_{m1} & c_{m2} & \dots & c_{mk} \end{pmatrix}, \tag{1.2}$$

где

$$c_{ij} = \sum_{t=1}^{z} a_{it} b_{tj} \quad (i = \overline{1,m}; j = \overline{1,k}) \tag{1.3}$$

Стандартный алгоритм умножения матриц реализует эту формулу.

1.2 Алгоритм Копперсмита-Винограда

В результате умножения двух прямоугольных матриц каждый элемент результирующей матрицы является скалярным произведением соответствующей строки и столбца исходных матриц.

Можно заметить, что часть этих вычислений, необходимых для скалярного произведения можно выполнить заранее, т.к. для каждой строки левой матрицы соответствующий столбец правой матрицы, на который скалярно умножается конкретная строка при вычислении столбца левой матрицы, не меняется.

Для удобства обозначений будем считать, что:
$\vec{U_i} = (u_1, ..., u_n)$ - i-ая строка матрицы $_{mn}$,
$\vec{V_j} = (v_1, ..., v_n)$ - j-ый столбец матрицы $_{nk}$.

Возьмем для примера $n = 4$.

Тогда $\vec{U_i} = (u_1, u_2, u_3, u_4)$, $\vec{V_j} = (v_1, v_2, v_3, v_4)$, и ячейка c_{ij} результирующей матрицы С будет описываться формулой (1.4):

$$c_{ij} = \vec{U_i} \cdot \vec{V_j} = u_1 \cdot v_1 + u_2 \cdot v_2 + u_3 \cdot v_3 + u_4 \cdot v_4 \tag{1.4}$$

Правую часть формулы (1.4) можно переписать в более сложной форме:

$$c_{ij} = \vec{U_i} \cdot \vec{V_j} = (u_1 + v_2) \cdot (u_2 + v_1) + (u_3 + v_4) \cdot (u_4 + v_3) - u_1 \cdot u_2 - u_3 \cdot u_4 - v_1 \cdot v_2 - v_3 \cdot v_4 \tag{1.5}$$

Это сделано из следующих соображений: несмотря на то, что вычисление формулы (1.5) требует большее число вычислений (6 умножений, 5 сложений и 4 вычитания) по сравнению с (1.4) (4 умножения и 4 сложения), преимуществм формулы (1.5) может стать то, что "хвост"формулы вычисляется заранее и далее используется повторно при умножении данной строки матрицы А на каждый столбец матрицы В. Соответственно, это позволит для каждой элемента выполнять лишь 2 умножения и 5 сложений, а как известно, операция сложения выполняется быстрее операции умножения в стандартных ЭВМ.

1.3 Вывод

В данном разделе были рассмотрены алгоритмы умножения прямоугольных матриц: стандартный алгоритм и алгоритм Копперсмита-Винограда. Основное отличие алгоритма Копперсмита-Винограда от стандартного - наличие предварительной обработки данных, что может дать выигрыш по производительности.

2 Конструкторский раздел

В данном разделе представлены схемы алгоритмов, описанных в аналитическом разделе, а также проведен анализ трудоемкости разработанных алгоритмов.

2.1 Схемы алгоритмов

На рисунке 2.1 представлена схема алгоритма стандартного умножения матриц.

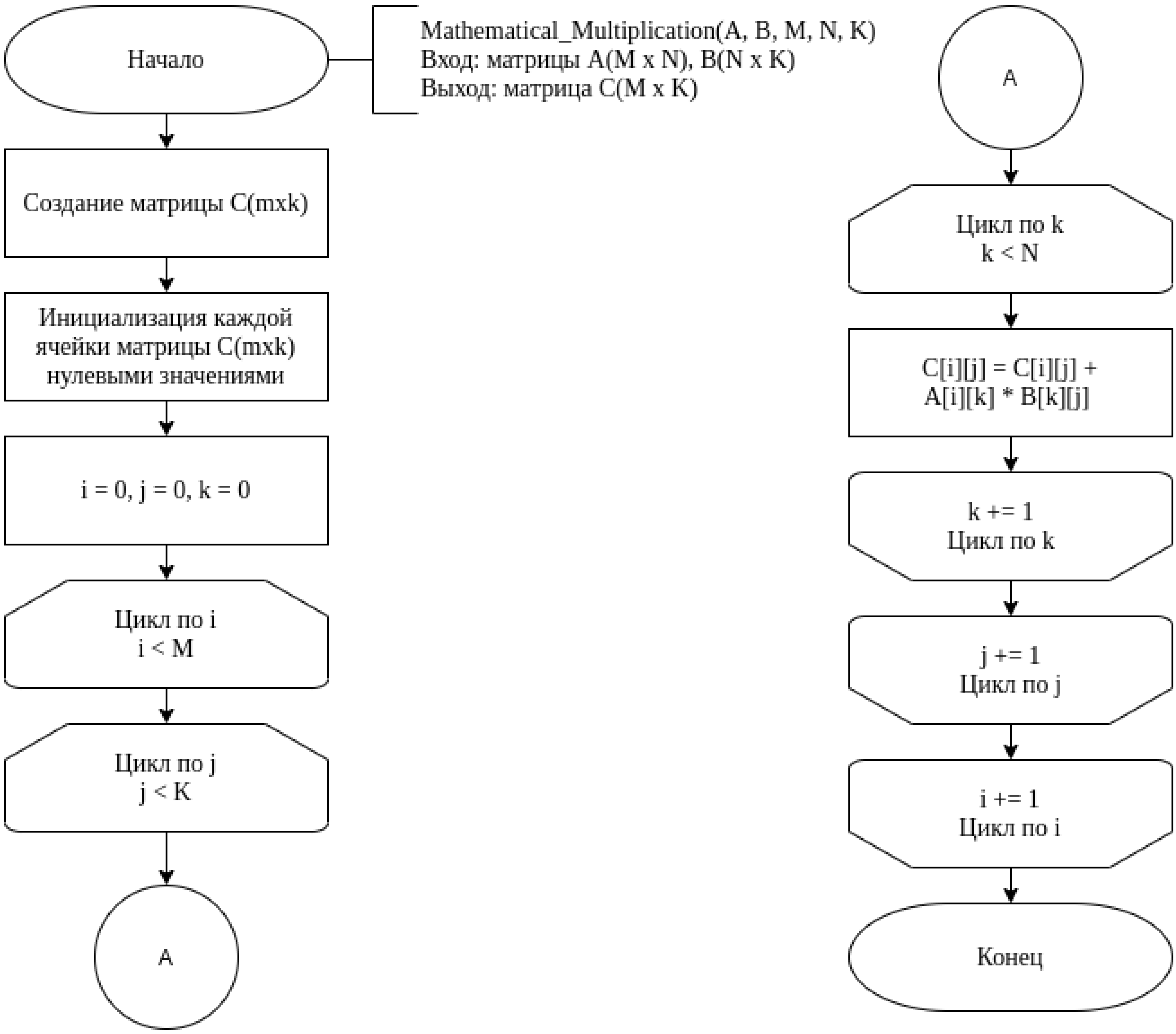


Рисунок 2.1 — Схема алгоритма стандартного умножения матриц

На рисунках 2.2-2.3 представлена схема алгоритма Копперсмита-Винограда умножения матриц.

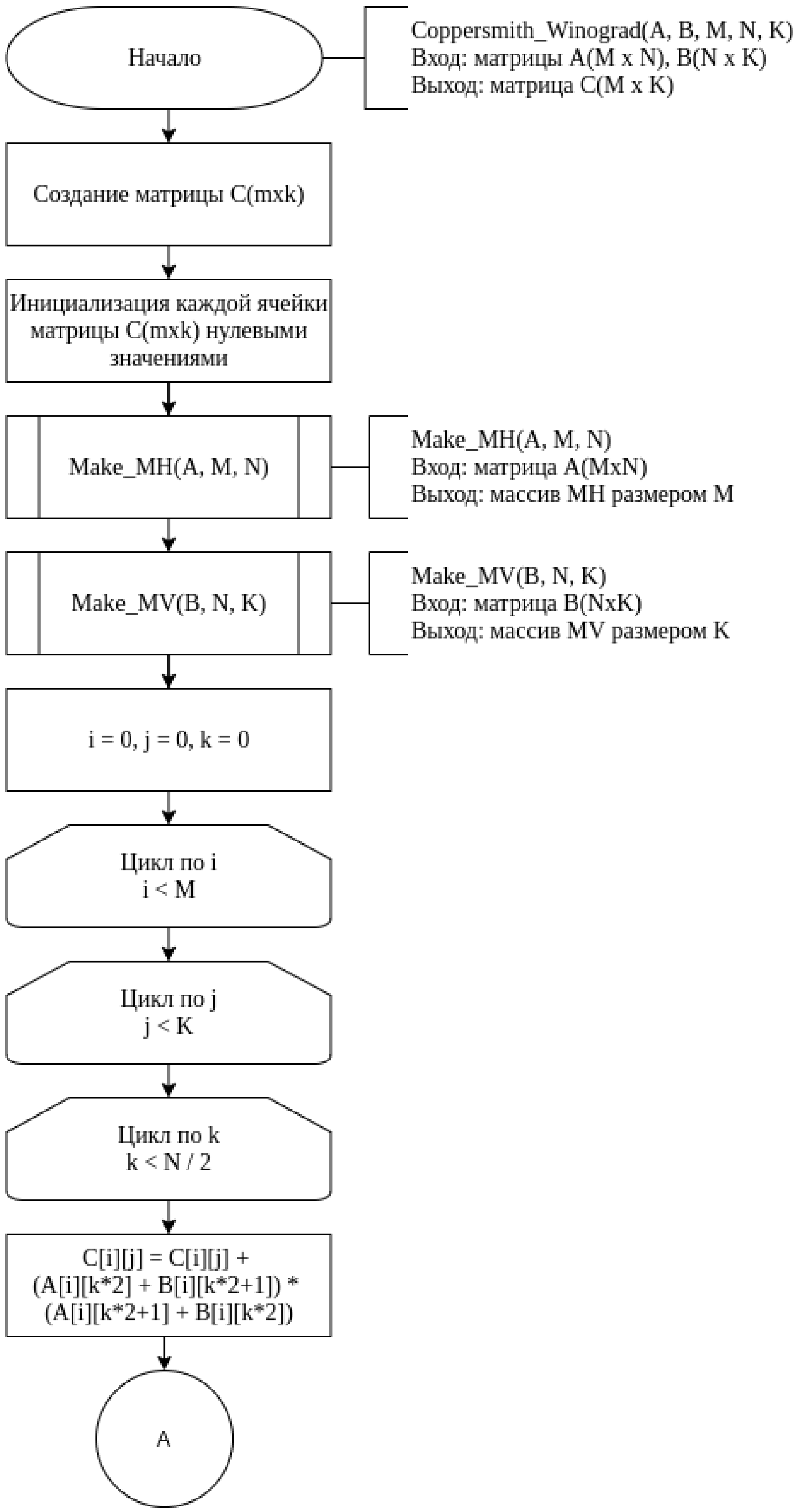


Рисунок 2.2 — Схема алгоритма Копперсмита-Винограда умножения матриц

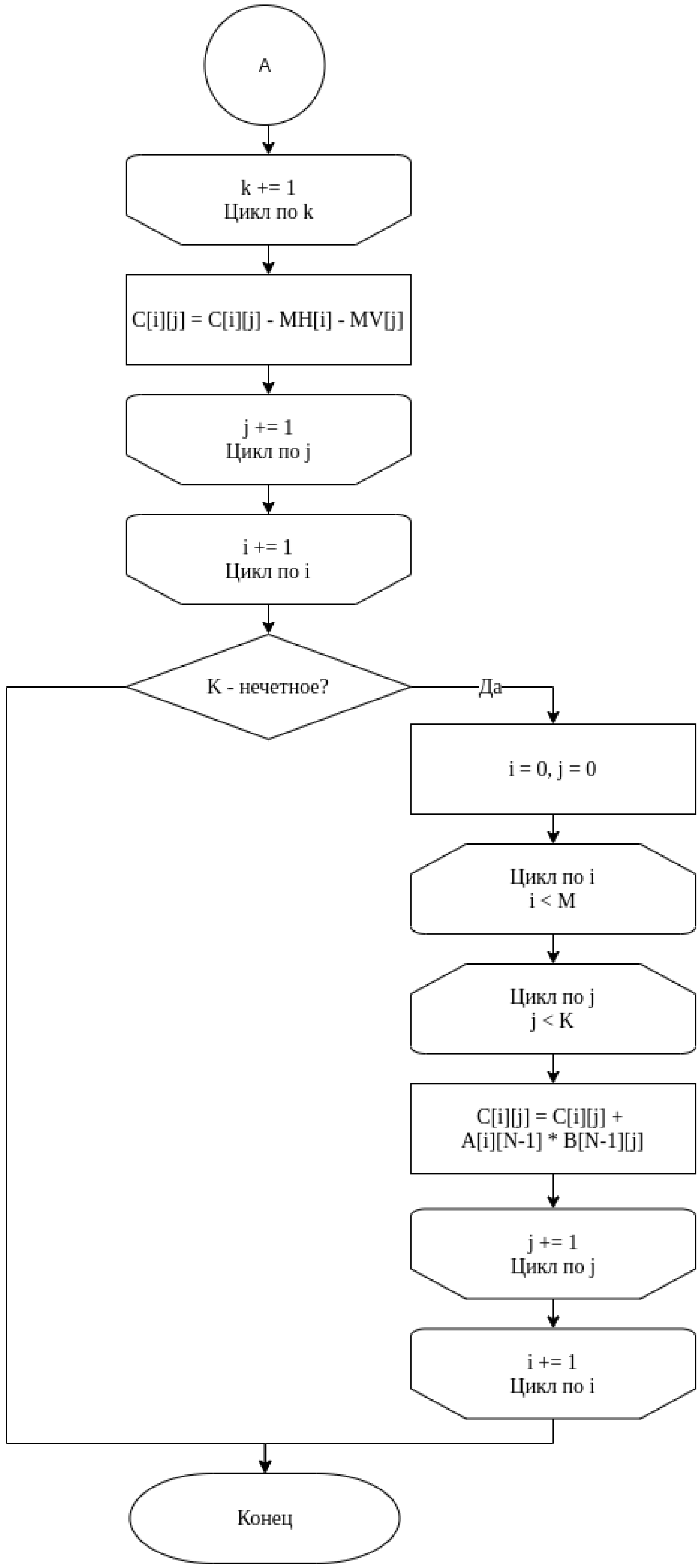


Рисунок 2.3 — Схема алгоритма Копперсмита-Винограда умножения матриц(продолжение)

На рисунке 2.4 представлены схемы функций заполнения массивов MH и MV, используемых в алгоритме Копперсмита-Винограда.

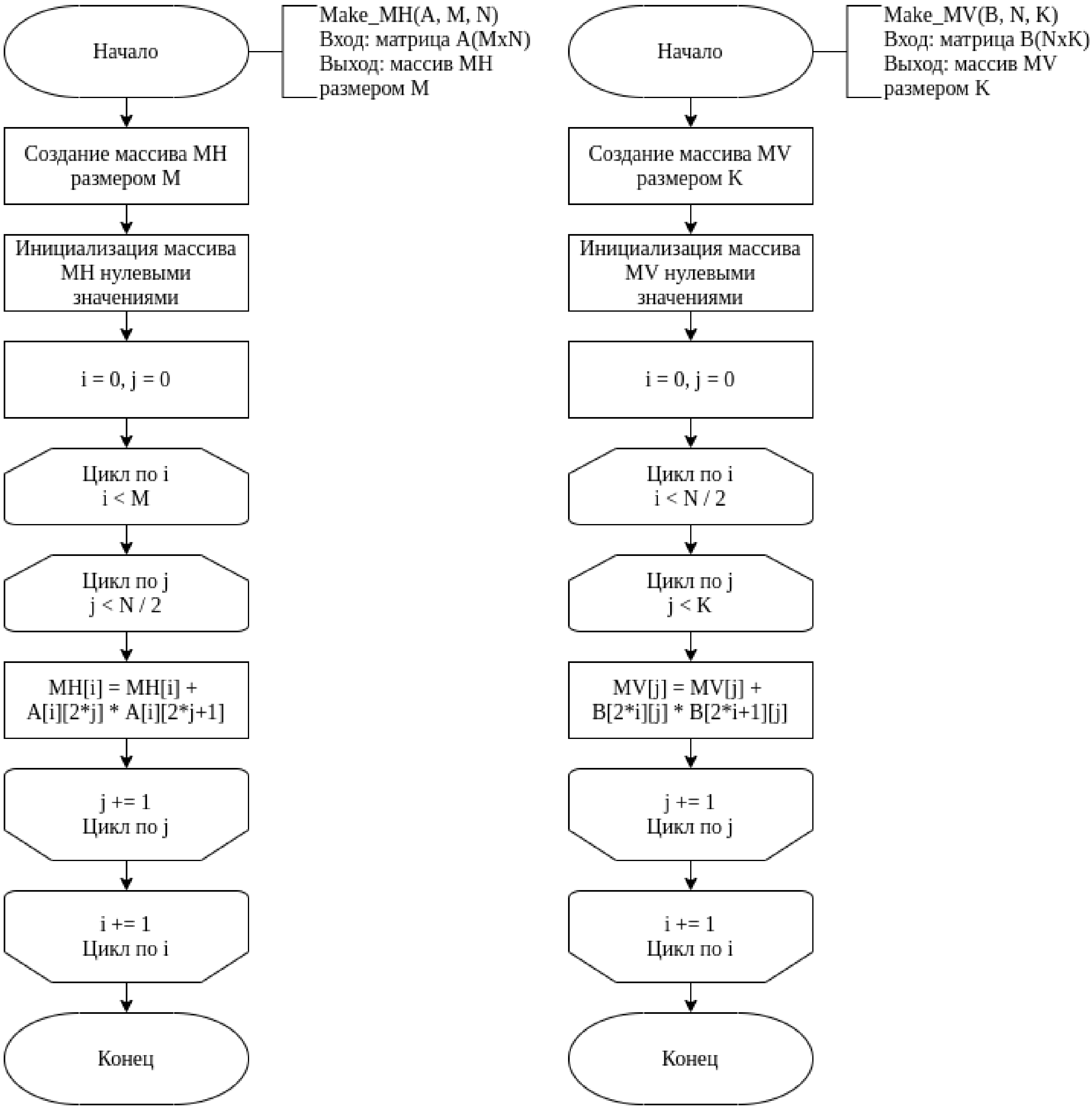


Рисунок 2.4 — Схемы функций заполнения массивов MH и MV, используемых в алгоритме Копперсмита-Винограда

На рисунках 2.5-2.6 представлена схема оптимизированного алгоритма Копперсмита-Винограда умножения матриц.

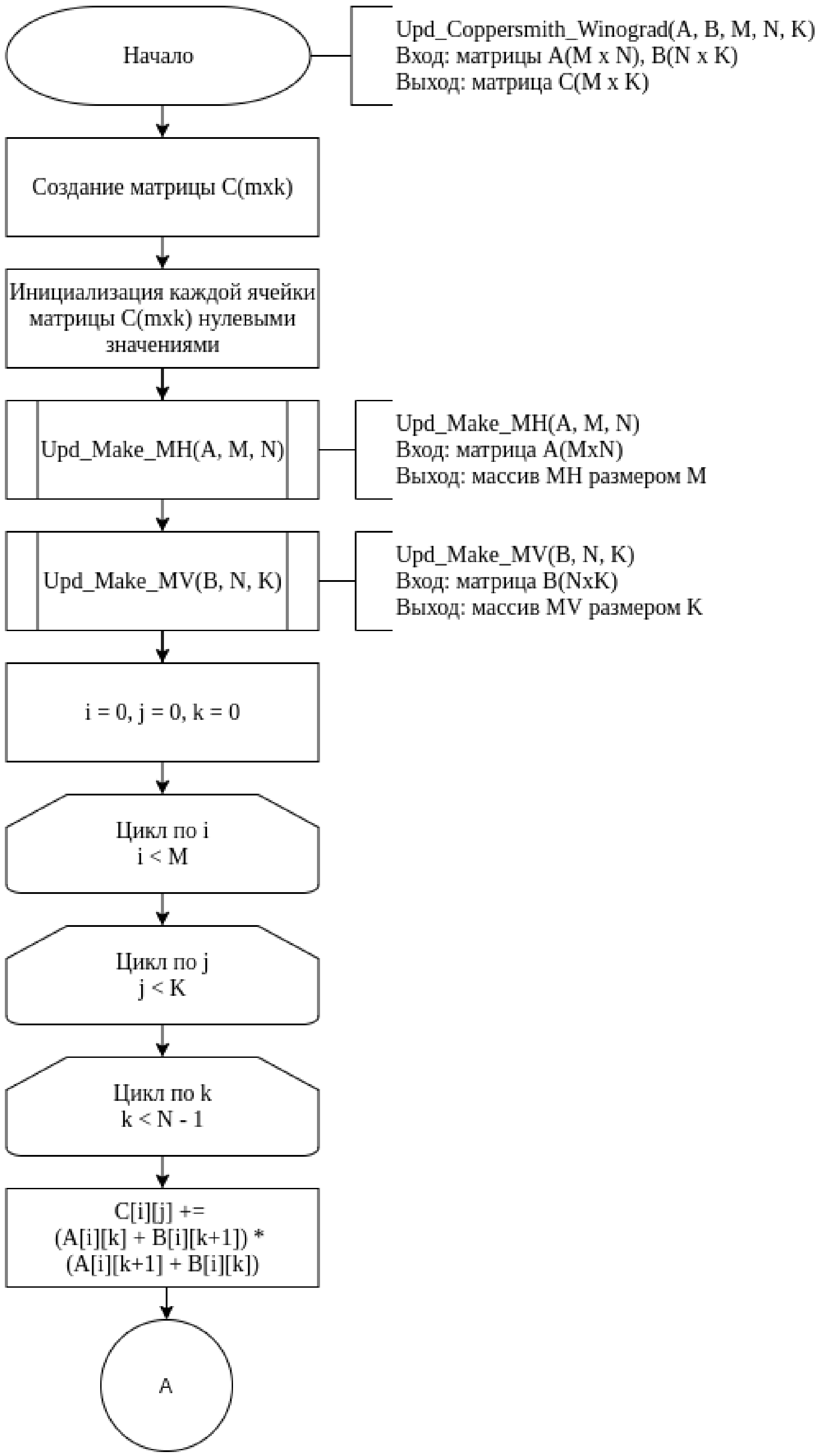


Рисунок 2.5 — Схема оптимизированного алгоритма Копперсмита-Винограда умножения матриц

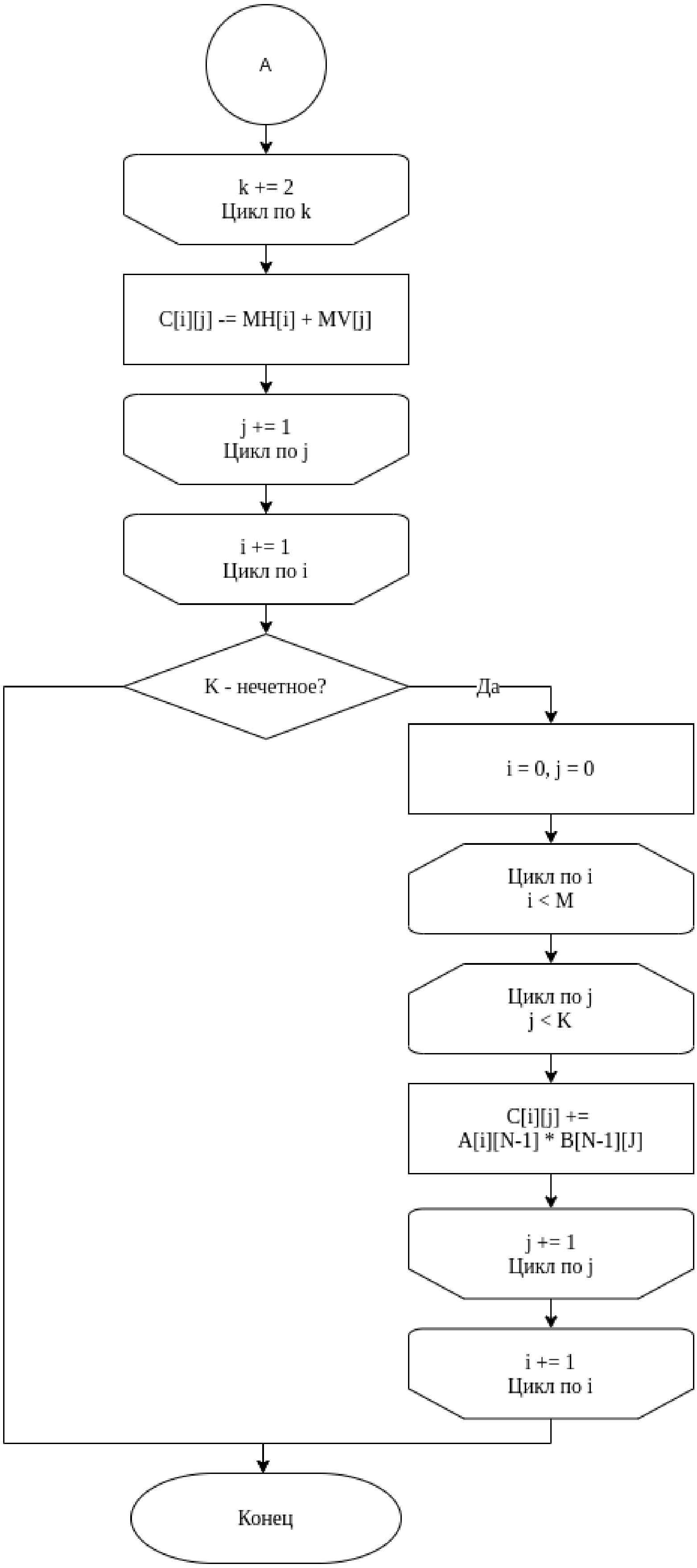


Рисунок 2.6 — Схема оптимизированного алгоритма Копперсмита-Винограда умножения матриц(продолжение)

На рисунке 2.7 представлены схемы функций заполнения массивов MH и MV, используемых в оптимизированном алгоритме Копперсмита-Винограда.

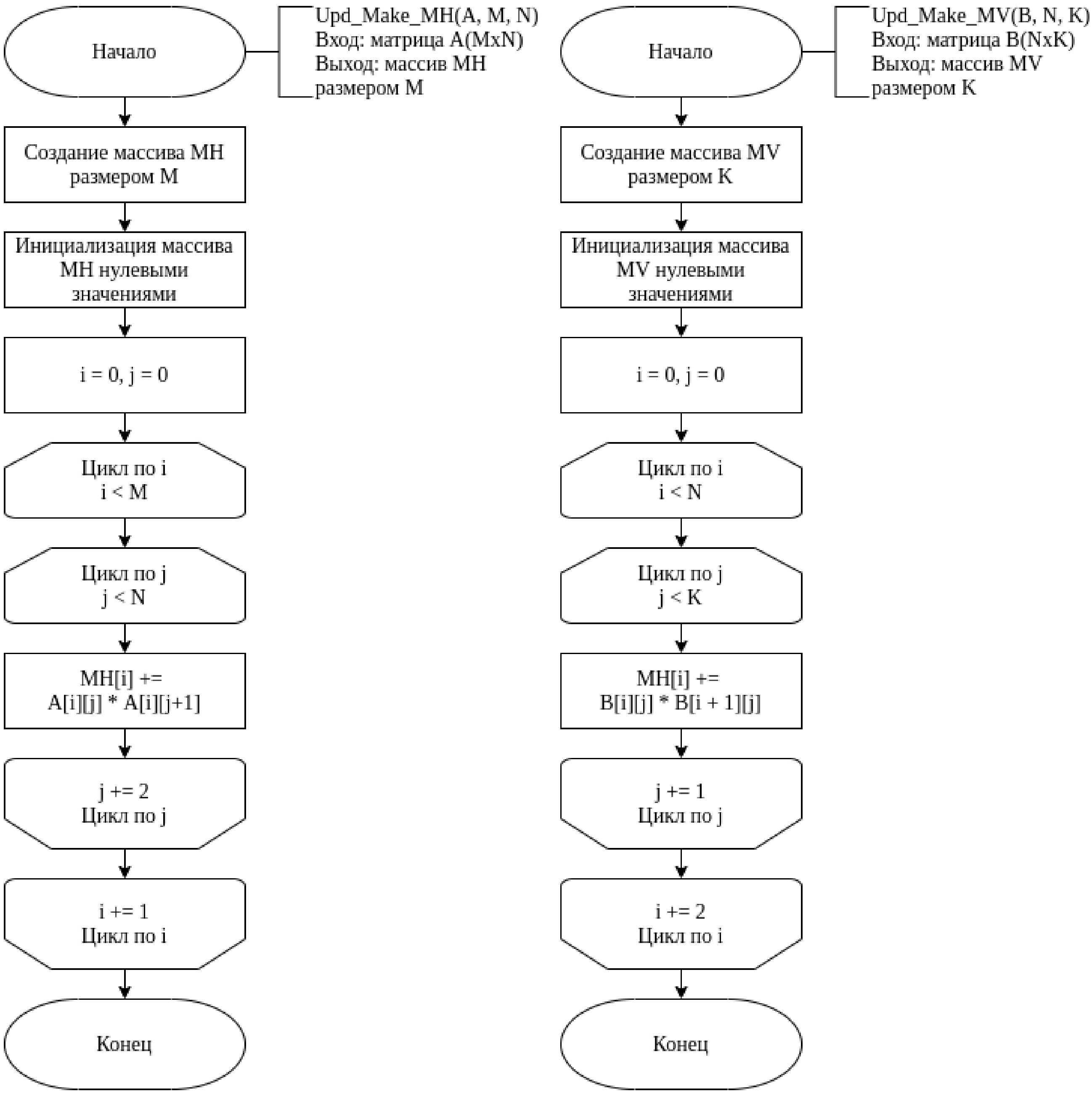


Рисунок 2.7 — Схемы функций заполнения массивов MH и MV, используемых в оптимизированном алгоритме Копперсмита-Винограда

Оптимизация Алгоритма Копперсмита-Винограда заключается в замене операции умножения на 2 в индексации в теле цикла на операцию $+=$ 2 в инкременте цикла. Также была произведена замена выражений вида $a = a + (...)$ на $a += (...)$.

2.2 Трудоемкость алгоритмов

Модель вычислений

Для последующего вычисления трудоемкости вводится следующая модель вычислений:

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

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

б) Операции из списка (2.2) имеют трудоемкость 2.

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

в) Трудоемкость оператора выбора if (условие) then (блок кода A) else (блок кода B) рассчитывается, как (2.3).

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

г) Трудоемкость цикла рассчитывается по формуле (2.4).

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

д) Трудоемкость вызова функции равна 0.

Примечание: во всех алгоритмах не рассматривается процесс создания матрицы, т.к. это действие присутствует во всех алгоритмах и не является самым трудоемким, потому что итоговая трудоемкость оценивается по наиболее быстро растущему слагаемому.

Оценка трудомекости алгоритмов проведена по схемам, представленным на рисунках 2.1-2.7 для алгоритмов стандартного умножения матриц, умножения матриц по Копперсмиту-Винограду и оптимизированной версии алгоритма Копперсмита-Винограда.

Трудоемкость стандартного алгоритма умножения матриц

Трудоемкость стандартного алгоритма умножения матриц состоит из:

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

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

— цикла по $k \in [1..N]$, трудоёмкость которого: $f = 2 + 14K$.

Трудоёмкость стандартного алгоритма равна трудоёмкости внешнего цикла:

$$f_{standard} = 2 + M \cdot (4 + N \cdot (4 + 14K)) = 2 + 4M + 4MN + 14MNK \approx 14MNK \qquad (2.5)$$

У стандартного алгоритма умножения матриц нет лучшего и худшего случая, т.к. нет ветвлей в схеме алгоритма.

Трудоемкость алгоритма Копперсмита-Винограда умножения матриц

Трудоёмкость алгоритма Копперсмита—Винограда состоит из:

а) создания и инициализации массивов MH и MV, трудоёмкость которого (2.13):

$$f_{init} = M + K + (2 + 4 \cdot M) + (2 + 4 \cdot K); \qquad (2.6)$$

б) заполнения массива MH, трудоёмкость которого (2.14):

$$f_{MH} = 2 + M \cdot (5 + 9 \cdot N); \qquad (2.7)$$

в) заполнения массива MV, трудоёмкость которого (2.15):

$$f_{MV} = 3 + \frac{N}{2} \cdot (5 + 17 \cdot K); \qquad (2.8)$$

г) цикла заполнения для чётных размеров, трудоёмкость которого (2.16):

$$f_{cycle} = 2 + M \cdot (13 + K \cdot (5 + 31 \cdot \frac{N}{2})); \qquad (2.9)$$

д) цикла, для дополнения умножения суммой последних нечётных строки и столбца, если общий размер нечётный, трудоемкость которого (2.17):

$$f_{last} = \begin{cases} 2, & \text{чётная,} \\ 4 + M \cdot (4 + 16K), & \text{иначе.} \end{cases} \qquad (2.10)$$

Итого, для худшего случая (нечётный общий размер матриц) имеем (2.18):

$$f = 15+27\cdot M+5\cdot K+\frac{5}{2}\cdot N+9\cdot MN+\frac{17}{2}NK+21\cdot MK+\frac{31}{2}\cdot MNK \approx 15.5\cdot MNK \tag{2.11}$$

Для лучшего случая (чётный общий размер матриц) имеем (2.19):

$$f = 13+23\cdot M+5\cdot K+\frac{5}{2}\cdot N+9\cdot MN+\frac{17}{2}NK+5\cdot MK+\frac{31}{2}\cdot MNK \approx 15.5\cdot MNK \tag{2.12}$$

Трудоемкость оптимизированного алгоритма Копперсмита-Винограда умножения матриц

Трудоёмкость оптимизированного алгоритма Копперсмита—Винограда состоит из:

а) создания и инициализации массивов MH и MV, трудоёмкость которого (2.13):

$$f_{init} = M + K + (2 + 4 \cdot M) + (2 + 4 \cdot K); \tag{2.13}$$

б) заполнения массива MH, трудоёмкость которого (2.14):

$$f_{MH} = 2 + M \cdot (14 + 11 \cdot \frac{N}{2}); \tag{2.14}$$

в) заполнения массива MV, трудоёмкость которого (2.15):

$$f_{MV} = 2 + \frac{N}{2} \cdot (4 + 11 \cdot K); \tag{2.15}$$

г) цикла заполнения для чётных размеров, трудоёмкость которого (2.16):

$$f_{cycle} = 2 + M \cdot (10 + K \cdot (5 + 10 \cdot N)); \tag{2.16}$$

д) цикла, для дополнения умножения суммой последних нечётных строки и столбца, если общий размер нечётный, трудоемкость которого (2.17):

$$f_{last} = \begin{cases} 2, & \text{чётная,} \\ 4 + M \cdot (4 + 13K), & \text{иначе.} \end{cases} \tag{2.17}$$

Итого, для худшего случая (нечётный общий размер матриц) имеем (2.18):

$$f = 14+33\cdot M+5\cdot K+2\cdot N+\frac{11}{2}\cdot MN+\frac{11}{2}\cdot NK+18\cdot MK+10\cdot MNK \approx 10\cdot MNK \tag{2.18}$$

Для лучшего случая (чётный общий размер матриц) имеем (2.19):

$$f = 12+29\cdot M+5\cdot K+2\cdot N+\frac{11}{2}\cdot MN+\frac{11}{2}\cdot NK+5\cdot MK+10\cdot MNK \approx 10\cdot MNK \tag{2.19}$$

2.3 Вывод

В данном разделе были разработаны блок-схемы для алгоритмов стандартного умножения матриц, умножения матриц по Копперсмиту-Винограду и оптимизированной версии алгоритма умножения матриц по Копперсмиту-Винограду. Также была произведена оценка представленных алгоритмов на основе полученных схем.

3 Технологический раздел

В данном разделе приведны требования к программному обеспечению, средства реализации и листинги кода.

3.1 Требования к ПО

К программе предъявляются следующие требования:

— На вход подаются размеры 2 матриц, которые выражаются целыми положительными числами, а также сами элементы матриц, которые выражаются целыми числами;

— Результат работы программы - матрица, которая является результатом умножения двух введенных ранее матриц; умножение требуется осуществить всеми тремя методами, схемы которых представлены в конструкторском разделе.

3.2 Средства реализации

В качестве языка программирования для реализации лабораторной работы был выбран язык C. Выбор этого языка обусловен его быстродействием и эффективностью, это легкочитаемый, лаконичный и гибкий язык. Также выбор обусловлен моим личным желанием получить больше практики написания программ на этом языке.

3.3 Листинг кода

В листинге 3.1 представлены пользовательские типы данных, которые используются в работе программы.

Листинг 3.1 — Пользовательские типы данных

```
typedef struct
{
    int **ptr;
    int dim1;
    int dim2;
} matrix_t;

typedef struct
{
```

```
    int *ptr;
    int size;
} vector_t;
```

В листинге 3.2 представлена реализация стандартного умножения матриц.

Листинг 3.2 — Реализация стандартного умножения матриц

```
matrix_t *math_mult(const matrix_t *A, const matrix_t *B)
{
    if (A->dim2 != B->dim1)
        return NULL;

    int **ptr = allocate_matrix(A->dim1, B->dim2);
    if (!ptr)
        return NULL;

    matrix_t *C = malloc(sizeof(matrix_t));
    if (!C)
        return NULL;
    fill(C, ptr, A->dim1, B->dim2);

    int i, j, k, M = A->dim1, N = A->dim2, K = B->dim2;
    for (i = 0; i < M; i++)
        for (j = 0; j < K; j++)
            (C->ptr)[i][j] = 0;

    for (i = 0; i < M; i++)
        for (j = 0; j < K; j++)
            for (k = 0; k < N; k++)
                (C->ptr)[i][j] = (C->ptr)[i][j] + (A->ptr)[i][k] *
                    (B->ptr)[k][j];
    return C;
}
```

В листинге 3.3 представлена реализация алгоритма Копперсмита-Винограда умножения матриц.

Листинг 3.3 — Алгоритм Копперсмита-Винограда

```
matrix_t *coppersmith_winograd(const matrix_t *A, const matrix_t *B)
{
    if (A->dim2 != B->dim1)
        return NULL;

    int **ptr = allocate_matrix(A->dim1, B->dim2);
    if (!ptr)
```

```
        return NULL;

    matrix_t *C = malloc(sizeof(matrix_t));
    if (!C)
    {
        free_matrix(ptr, A->dim1);
        return NULL;
    }
    fill(C, ptr, A->dim1, B->dim2);

    int i, j, k, M = A->dim1, N = A->dim2, K = B->dim2;

    vector_t *mh = malloc(sizeof(vector_t));
    if (!mh)
    {
        free(C);
        return NULL;
    }

    vector_t *mv = malloc(sizeof(vector_t));
    if (!mv)
    {
        free(mh);
        free(C);
        return NULL;
    }

    mh = make_mh(A);
    if (!mh)
    {
        free(mv);
        free(C);
        return NULL;
    }
    mv = make_mv(B);
    if (!mv)
    {
        free(mh);
        free(C);
        return NULL;
    }

    for (i = 0; i < M; i++)
        for (j = 0; j < K; j++)
            (C->ptr)[i][j] = 0;

```

```
    for (i = 0; i < M; i++)
        for (j = 0; j < K; j++)
        {
            for (k = 0; k < N / 2; k++)
            {
                (C->ptr)[i][j] = (C->ptr)[i][j] +
                                      ((A->ptr)[i][k * 2] + (B->ptr)[k * 2 + 1][j]) *
                                      ((A->ptr)[i][k * 2 + 1] + (B->ptr)[k * 2][j]);
            }
            (C->ptr)[i][j] = (C->ptr)[i][j] - (mh->ptr)[i] - (mv->ptr)[j];
        }

    if (N % 2 == 1)
    {
        for (i = 0; i < M; i++)
            for (j = 0; j < K; j++)
                (C->ptr)[i][j] = (C->ptr)[i][j] + (A->ptr)[i][N - 1] * (B->ptr)[N - 1][j];
    }

    free(mh);
    free(mv);

    return C;
}

vector_t *make_mh(const matrix_t *A)
{
    int M = A->dim1, N = A->dim2;
    int *ptr = malloc(sizeof(int) * M);
    if (!ptr)
        return NULL;

    vector_t *mh = malloc(sizeof(vector_t));
    if (!mh)
        return NULL;
    mh->ptr = ptr;
    mh->size = M;

    for (int i = 0; i < M; i++)
        (mh->ptr)[i] = 0;

    for (int i = 0; i < M; i++)
        for (int j = 0; j < N / 2; j++)
```

```
            (mh->ptr)[i] = (mh->ptr)[i] + (A->ptr)[i][2 * j] * (A->ptr)[i][2 * j + 1];

    return mh;
}

vector_t *make_mv(const matrix_t *B)
{
    int N = B->dim1, K = B->dim2;
    int *ptr = malloc(sizeof(int) * K);
    if (!ptr)
        return NULL;

    vector_t *mv = malloc(sizeof(vector_t));
    if (!mv)
        return NULL;
    mv->ptr = ptr;
    mv->size = K;

    for (int i = 0; i < K; i++)
        (mv->ptr)[i] = 0;

    for (int i = 0; i < N / 2; i++)
        for (int j = 0; j < K; j++)
            (mv->ptr)[j] = (mv->ptr)[j] + (B->ptr)[2 * i][j] * (B->ptr)[2 * i + 1][j];

    return mv;
}
```

В листинге 3.4 представлена реализация оптимизированного алгоритма Копперсмита-Винограда умножения матриц.

Листинг 3.4 — Оптимизированный алгоритм Копперсмита-Винограда

```
matrix_t *upd_coppersmith_winograd(const matrix_t *A, const matrix_t *B)
{
    if (A->dim2 != B->dim1)
        return NULL;

    int **ptr = allocate_matrix(A->dim1, B->dim2);
    if (!ptr)
        return NULL;

    matrix_t *C = malloc(sizeof(matrix_t));
    if (!C)
    {
```

```
            free_matrix(ptr, A->dim1);
            return NULL;
        }
        fill(C, ptr, A->dim1, B->dim2);

        int i, j, k, M = A->dim1, N = A->dim2, K = B->dim2;

        vector_t *mh = malloc(sizeof(vector_t));
        if (!mh)
        {
            free(C);
            return NULL;
        }

        vector_t *mv = malloc(sizeof(vector_t));
        if (!mv)
        {
            free(mh);
            free(C);
            return NULL;
        }

        mh = make_mh(A);
        if (!mh)
        {
            free(mv);
            free(C);
            return NULL;
        }
        mv = make_mv(B);
        if (!mv)
        {
            free(mh);
            free(C);
            return NULL;
        }

        for (i = 0; i < M; i++)
            for (j = 0; j < K; j++)
                (C->ptr)[i][j] = 0;

        for (i = 0; i < M; i++)
            for (j = 0; j < K; j++)
            {
                for (k = 0; k < N - 1; k+=2)
                {
```

```
                (C->ptr)[i][j] += ((A->ptr)[i][k] + (B->ptr)[k + 1][j]) *
                                  ((A->ptr)[i][k + 1] + (B->ptr)[k][j]);
            }
            (C->ptr)[i][j] -= (mh->ptr)[i] + (mv->ptr)[j];
        }

    if (N % 2 == 1)
    {
        for (i = 0; i < M; i++)
            for (j = 0; j < K; j++)
                (C->ptr)[i][j] += (A->ptr)[i][N - 1] * (B->ptr)[N - 1][j];
    }

    free(mh);
    free(mv);

    return C;
}

vector_t *upd_make_mh(const matrix_t *A)
{
    int M = A->dim1, N = A->dim2;
    int *ptr = malloc(sizeof(int) * M);
    if (!ptr)
        return NULL;

    vector_t *mh = malloc(sizeof(vector_t));
    if (!mh)
        return NULL;
    mh->ptr = ptr;
    mh->size = M;

    for (int i = 0; i < M; i++)
        (mh->ptr)[i] = 0;

    for (int i = 0; i < M; i++)
        for (int j = 0; j < N; j+=2)
            (mh->ptr)[i] += (A->ptr)[i][j] * (A->ptr)[i][j + 1];

    return mh;
}

vector_t *upd_make_mv(const matrix_t *B)
{
    int N = B->dim1, K = B->dim2;
    int *ptr = malloc(sizeof(int) * K);
```

```
    if (!ptr)
        return NULL;

    vector_t *mv = malloc(sizeof(vector_t));
    if (!mv)
        return NULL;
    mv->ptr = ptr;
    mv->size = K;

    for (int i = 0; i < K; i++)
        (mv->ptr)[i] = 0;

    for (int i = 0; i < N; i+=2)
        for (int j = 0; j < K; j++)
            (mv->ptr)[j] += (B->ptr)[i][j] * (B->ptr)[i + 1][j];

    return mv;
}
```

В листинге 3.5 представлены вспомогательные функции.

Листинг 3.5 — Вспомогательные функции

```
int **allocate_matrix(int m, int n)
{
    int **data = calloc(m, sizeof(int*));
    if (!data)
        return NULL;
    for (int i = 0; i < m; i++)
    {
        data[i] = malloc(n * sizeof(int));
        if (!data[i])
        {
            free_matrix(data, m);
            return NULL;
        }
    }
    return data;
}

void fill(matrix_t *M, int **ptr, int a, int b)
{
    M->ptr = ptr;
    M->dim1 = a;
    M->dim2 = b;
}

void free_matrix(int **matrix, int size)
```

```
{
    for (int i = 0; i < size; i++)
        free(matrix[i]);
    free(matrix);
}
```

3.4 Вывод

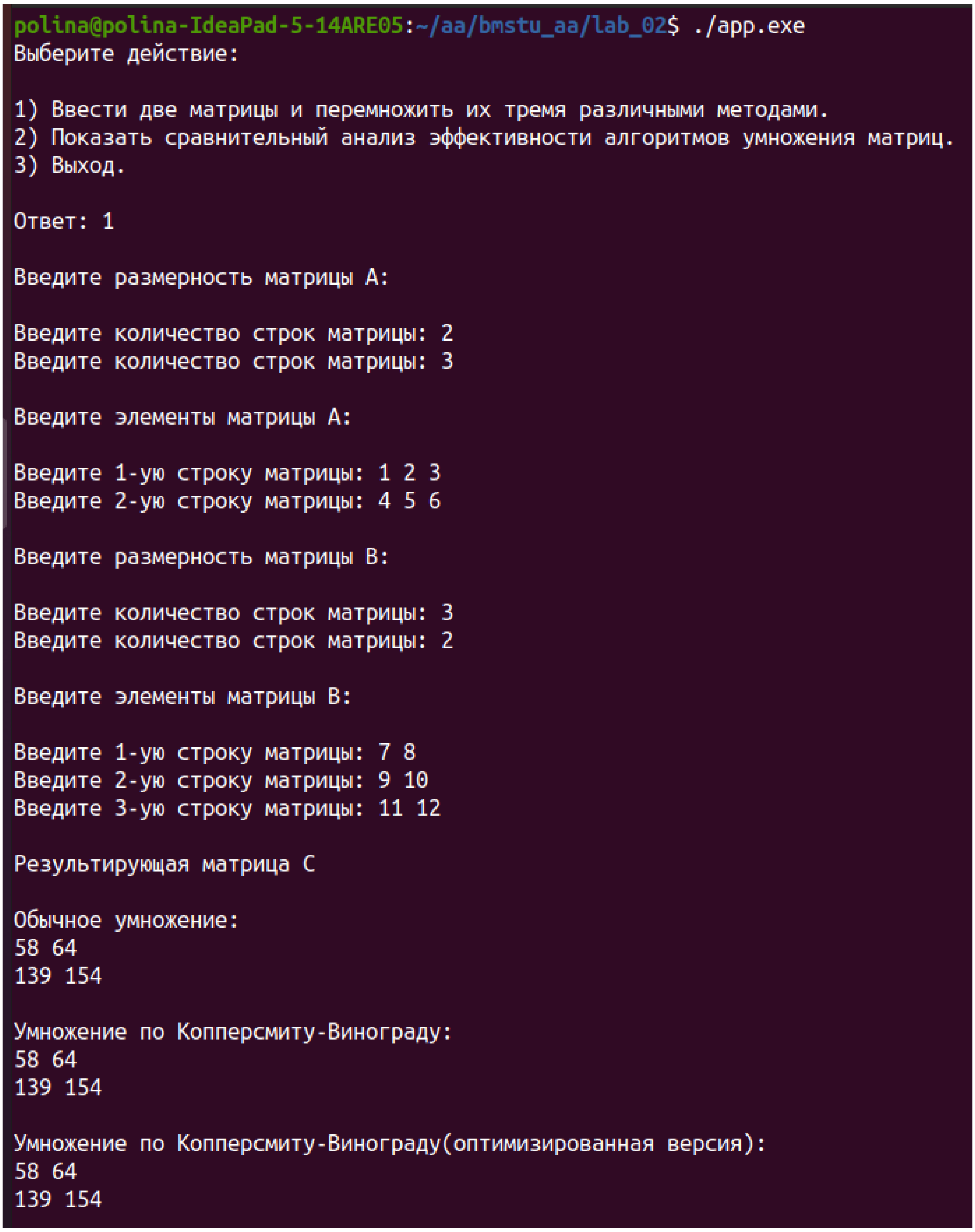
В данном разделе были реализованы выбранные алгоритмы умножения матриц: стандартный алгоритм, алгоритм Копперсмита-Винограда, а также его оптимизированная версия.

4 Экспериментальный раздел

В данном разделе представлен пользовательский интерфейс, а также проведена оценка эффективности алгоритмов.

4.1 Пример работы

На рисунке 4.1 приведен пример работы программы.



```
polina@polina-IdeaPad-5-14ARE05:~/aa/bmstu_aa/lab_02$ ./app.exe
Выберите действие:

1) Ввести две матрицы и перемножить их тремя различными методами.
2) Показать сравнительный анализ эффективности алгоритмов умножения матриц.
3) Выход.

Ответ: 1

Введите размерность матрицы A:

Введите количество строк матрицы: 2
Введите количество строк матрицы: 3

Введите элементы матрицы A:

Введите 1-ую строку матрицы: 1 2 3
Введите 2-ую строку матрицы: 4 5 6

Введите размерность матрицы B:

Введите количество строк матрицы: 3
Введите количество строк матрицы: 2

Введите элементы матрицы B:

Введите 1-ую строку матрицы: 7 8
Введите 2-ую строку матрицы: 9 10
Введите 3-ую строку матрицы: 11 12

Результирующая матрица C

Обычное умножение:
58 64
139 154

Умножение по Копперсмиту-Винограду:
58 64
139 154

Умножение по Копперсмиту-Винограду(оптимизированная версия):
58 64
139 154
```

Рисунок 4.1 — Пример работы программы

4.2 Тестовые данные

В таблице 4.1 приведены тестовые данные, на которых было протестировано разработанное ПО. Все тесты были успешно пройдены.

Таблица 4.1 — Таблица тестовых данных

Матрица 1	Матрица 2	Результат
$\begin{pmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \\ 1 & 2 & 3 \end{pmatrix}$	$\begin{pmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \\ 1 & 2 & 3 \end{pmatrix}$	$\begin{pmatrix} 6 & 12 & 18 \\ 6 & 12 & 18 \\ 6 & 12 & 18 \end{pmatrix}$
$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$	$\begin{pmatrix} 7 & 8 \\ 9 & 10 \\ 11 & 12 \end{pmatrix}$	$\begin{pmatrix} 58 & 64 \\ 139 & 154 \end{pmatrix}$
$\begin{pmatrix} -1 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 0 & 2 & 4 \\ 6 & 8 & 10 \\ 12 & 14 & 16 \end{pmatrix}$	$\begin{pmatrix} 12 & 12 & 12 \end{pmatrix}$
$\begin{pmatrix} 0 & 2 & 4 \\ 6 & 8 & 10 \\ 12 & 14 & 16 \end{pmatrix}$	$\begin{pmatrix} -1 \\ 0 \\ 1 \end{pmatrix}$	$\begin{pmatrix} 4 \\ 4 \\ 4 \end{pmatrix}$
$\begin{pmatrix} 7 \end{pmatrix}$	$\begin{pmatrix} 10 \end{pmatrix}$	$\begin{pmatrix} 70 \end{pmatrix}$
$\begin{pmatrix} 1 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 1 \end{pmatrix}$	Невозможно произвести умножение матриц

4.3 Технические характеристики

Технические характеритиски машины, на которой выполнялось тестирование:

— Операционная система: Ubuntu[2] Linux[3] 20.04 64-bit.

— Оперативная память: 16 Gb.

— Процессор: AMD(R) Ryzen(TM)[4] 5 4500U CPU @ 2.3 CHz

4.4 Время выполнения алгоритмов

Время выполнения агоритмов (процессорное) замерялось с помощью ассемблерной вставки, которая ведет посчет тиков процессора[5]:

Листинг 4.1 — "Ассемблерная вставка для замера тиков процессора"

```
uint64_t tick(void)
{
    uint32_t high, low;
    __asm__ __volatile__(
        "rdtsc\n"
        "movl %%edx, %0\n"
        "movl %%eax, %1\n"
        : "=r"(high), "=r"(low)::"%rax", "%rbx", "%rcx", "%rdx");

    uint64_t ticks = ((uint64_t)high << 32) | low;

    return ticks;
}
```

В таблице 4.2 представлены замеры процессорного времени тиков для различных размеров матриц с нечетным общим размером.

Таблица 4.2 — Таблица замеров процессорного времени (в тиках) для матриц с нечетным общим размером.

Размерность	С	К-В	О-К-В
101	17 037 910	13 100 466	12 823 225
201	131 847 305	102 725 496	101 244 779
301	407 783 809	311 760 957	306 685 575
401	639 489 516	489 074 549	485 526 010
501	1 256 029 866	961 576 809	952 782 704
601	2 242 731 048	1 692 616 669	1 678 574 307
701	3 560 262 443	2 674 993 062	2 620 174 437
801	5 485 556 615	4 069 537 565	4 008 247 415
901	7 791 519 909	5 823 951 720	5 772 267 059
1001	10 289 365 483	7 834 348 343	7 714 377 364

Где С - стандартный алгоритм умножения матриц, К-В - алгоритм Коперсмита-Винограда, О-К-В - оптимизированный алгоритм Копперсмита-Винограда.

В таблице 4.2 представлены замеры процессорного времени тиков для различных размеров матриц с четным общим размером.

Таблица 4.3 — Таблица замеров процессорного времени (в тиках) для матриц с четным общим размером.

Размерность	С	К-В	О-К-В
100	10 623 597	8 225 198	8 087 864
200	81 324 284	63 580 094	62 517 839
300	270 604 717	20 9040 189	206 992 625
400	642 838 850	494 831 428	489 359 653
500	12 6969 4362	971 644 925	960 102 228
600	2 182 536 000	1 662 890 236	1 645 896 353
700	3 503 856 091	266 001 6728	2 634 365 050
800	5 605 076 609	4 117 490 507	3 974 643 328
900	8 151 095 988	6 003 855 039	5 814 001 428
1000	1 227 732 0628	8 989 576 451	8 871 964 545

На рисунке 4.2 показана зависимость процессорного времени для матриц с нечетным общим размером.

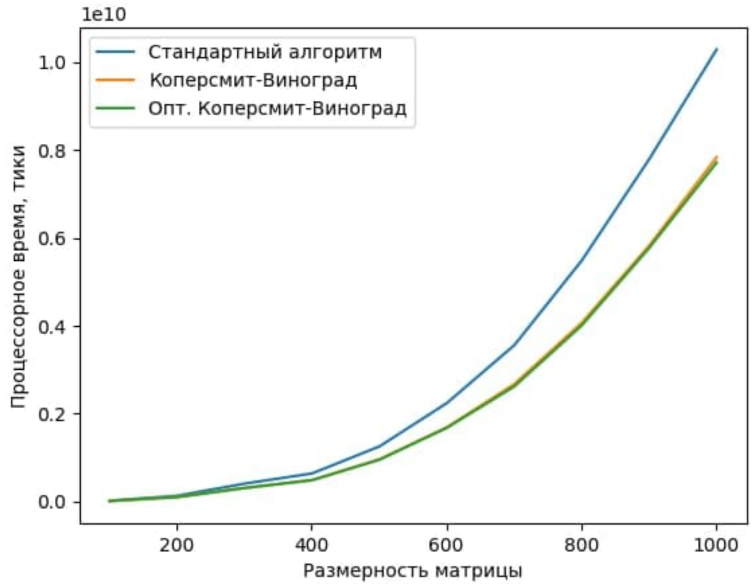


Рисунок 4.2 — Зависимость процессорного времени для матриц с нечетным общим размером

На рисунке 4.3 показана зависимость процессорного времени для матриц с четным общим размером.

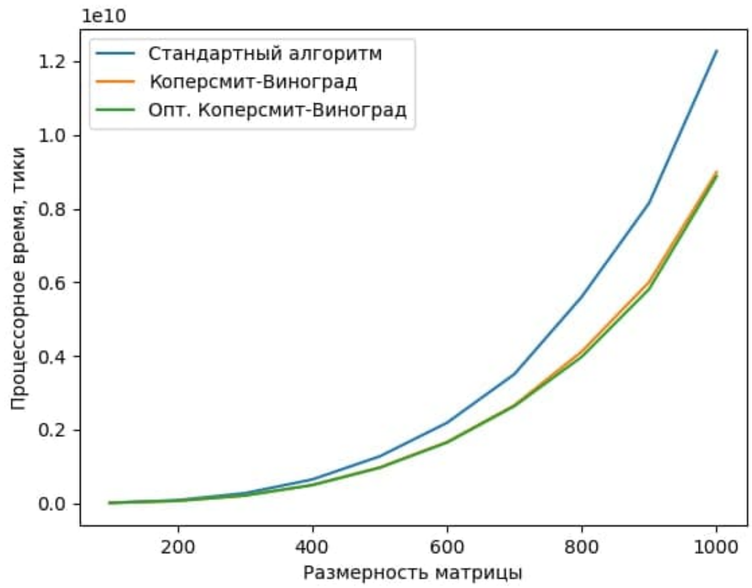


Рисунок 4.3 — Зависимость процессорного времени для матриц с четным общим размером

4.5 Вывод

В данном разделе было проведено тестирование программы, а также замеры процессорного времени работы алгоритмов умножения матриц.

В результате замеров был сделан вывод, что самый эффективный среди трех рассмотренных - стандартный алгоритм умножения матриц. Как и ожидалось, оптимизированная версия алгоритма Копперсмита-Винограда работает быстрее изначальной версии этого алгоритма, однако выигрыш не слишком большой - порядка 5-7%.

Также было подтверждено замедление работы алгоритмов в случае нечетного общего размера матриц.

Заключение

В ходе выполнения лабораторной работы были выполнены поставленные задачи, а именно:

а) Были рассмотрены и изучены следующие алгоритмы умножения матриц: классический метод и метод Копперсмита-Винограда.

б) Были реализованы выбранные методы умножения матриц.

в) Была рассчитана трудоемкость методов умножения матриц.

г) Были сравнены временные характеристики экспериментально.

д) На основании проделанной работы были сделаны выводы.

Экспериментально были подтверждены различия в производительности различных методов умножения матриц: Самый эффективный среди трех рассмотренных - стандартный алгоритм умножения матриц. Как и ожидалось, оптимизированная версия алгоритма Копперсмита-Винограда работает быстрее изначальной версии этого алгоритма, однако выигрыш не слишком большой - порядка 5-7%.

Также было подтверждено замедление работы алгоритмов в случае нечетного общего размера матриц.

Таким образом, выбирая алгоритм умножения матриц, следует по возможности отдать предпочтение алгоритму Копперсмита-Винограда.

Список использованных источников

1. Умножение матриц. — [Электронный ресурс]. Режим доступа: http://www.algolib.narod.ru/Math/Matrix.html., 01.10.2021.

2. Ununtu. — [Электронный ресурс]. Режим доступа: https://ru.wikipedia.org/wiki/Ubuntu., 16.09.2021.

3. Linux. — [Электронный ресурс]. Режим доступа: https://ru.wikipedia.org/wiki/Linux., 16.09.2021.

4. Процессор AMD Ryzen(TM) 5. — [Электронный ресурс]. Режим доступа: https://shop.lenovo.ru/product/81YM007FRU/., 21.09.2021.

5. C/C++: как измерять процессорное время. — [Электронный ресурс]. Режим доступа: https://habr.com/ru/post/282301/., 16.09.2021.