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»

Лабораторная работа №2
по дисциплине «Анализ Алгоритмов»

Тема Алгоритмы умножения матриц

Студент Куликов Е. А.

Группа ИУ7-56Б

Преподаватель Волкова Л. Л.

Москва — 2024 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Умножение матриц — одна из основных операций над матрицами. Матрица, получаемая в результате операции умножения, называется произведением матриц. Матричное умножение определено только тогда, когда число столбцов первой матрицы равно числу строк второй матрицы [1].

Целью данной лабораторной работы является анализ и реализация различных алгоритмов умножения матриц, таких как классический алгоритм и алгоритм Винограда. Для достижения поставленной цели необходимо решить следующие задачи:

— проанализировать классический алгоритм умножения матриц и алгоритм Винограда;

— реализовать указанные алгоритмы;

— провести сравнение эффективности данных алгоритмов по времени выполнения на плате STM;

— описать и обосновать полученные результаты в отчете о выполненной лабораторной работе.

1 Аналитическая часть

В данном разделе будут рассмотрены классический алгоритм и алгоритм Винограда для умножения матриц.

1.1 Классический алгоритм умножения матриц

Первый способ умножения матриц — классический. Пусть есть матрица A размером $n \times m$ с элементами a_{ij} и матрица B размером $m \times k$ с элементами b_{ij} (число строк второй матрицы должно быть равно числу столбцов первой матрицы). Тогда результатом умножения этих матриц будет являться матрица C размером $n \times k$ с элементами:

$$c_{ij} = \sum_{k=1}^{n} a_{ik} \cdot b_{kj} \tag{1.1}$$

1.2 Алгоритм Винограда

Алгоритм Винограда — попытка оптимизировать классический алгоритм умножения матриц путем снижения доли «дорогих» операций, в частности — умножения.

Классическое умножение строки на столбец:

$$c_{ij} = \begin{pmatrix} u_1 & u_2 & u_3 & u_4 \end{pmatrix}_i \times \begin{pmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \end{pmatrix}_j = u_1 \cdot v_1 + u_2 \cdot v_2 + u_3 \cdot v_3 + u_4 \cdot v_4 \tag{1.2}$$

На 4 элемента приходится 2 умножения. В алгоритме Винограда формула изменена:

$$c_{ij} = (u_1 + v_2) \cdot (u_2 + v_1) + (u_3 + v_4) \cdot (u_4 + v_3) - u_1 \cdot u_2 - u_3 \cdot u_4 - v_1 \cdot v_2 - v_3 \cdot v_4 \tag{1.3}$$

Для первых двух компонент со скобками получаем 1 умножение на 4 элемента. При этом третью ки четвертую компоненту можно вычислить заранее и использовать для умножения данной строки матрицы A на все столбцы матрицы B или данного столбца матрицы B на все строки матрицы A. Эти значения накапливаются в отдельных массивах — $MulH$ и $MulV$.

2 Конструкторская часть

В данном разделе будут разработаны классический алгоритм и алгоритм Винограда умножения матриц.

2.1 Модель вычислений

Для описания трудоемкости алгоритмов вводится модель вычислений:

— трудоемкость базовых операций =, +, -, +=, -=, ==, !=, <, >, <=, >=, [], «, » принимается единичной, трудоемкость операций *, /, *=, /=, %, %= принимается равной 2;

— трудоемкость оператора цикла вида $for([init];[comp];[inc])\{[body]\}$ при известных трудоемкостях его составных частей $f_{init}, f_{comp}, f_{inc}, f_{body}$ принимается за $f_{cycle} = f_{init} + f_{comp} + M \cdot (f_{body} + f_{inc} + f_{comp})$ где M — количество шагов цикла;

— трудоемкость условного перехода принимается равной нулю, для условного оператора вида $if[condition]\{[code1]\}else\{[code2]\}$ при известных трудоемкостях $f_{code1}, f_{code2}, f_{condition}$ трудоемкость оценивается как

$$f = \begin{cases} f_{condition} + min(f_{code1}, f_{code2}), & \text{для лучшего случая} \\ f_{condition} + max(f_{code1}, f_{code2}), & \text{для худшего случая} \end{cases} \tag{2.1}$$

2.2 Оценки трудоемкости алгоритмов

2.2.1 Классический алгоритм

Трудоемкость классического алгоритма умножения матриц:

Листинг 1 – Классический алгоритм умножения матриц

```
for (int i = 0; i < M; i++)
        for (int j = 0; j < Q; j++){
            C[i][j] = 0;    //3
            for (int k = 0; k < N; k++)
                C[i][j] = C[i][j] + A[i][k] * B[k][j];
        }
```

Трудоемкость алгоритма равна:

$$f_{classical} = 14 \cdot M \cdot N \cdot Q + 7 \cdot Q + 4 \cdot M + 2 \tag{2.2}$$

2.2.2 Алгоритм Винограда

Трудоемкость алгоритма Винограда:

Листинг 2 – Алгоритм умножения матриц Винограда

```
// I - filling of MulH
for (int i = 0; i < M; i++){
    for (int k = 0; k < N / 2; k++){
        MulH[i] = MulH[i] + A[i][k * 2] * A[i][k * 2 + 1];
    }
}
// II - filling of MulV
for (int i = 0; i < Q; i++){
    for (int k = 0; k < N / 2; k++){
        MulV[i] = MulV[i] + B[k * 2][i] * B[k * 2 + 1][i];
    }
}
// III - filling of C
for (int i = 0; i < M; i++) {
    for (int j = 0; j < Q; j++){
        C[i][j] = -MulH[i] - MulV[j];
        for (int k = 0; k < N / 2; k++){
            C[i][j] = C[i][j] + (A[i][k * 2 + 1] + B[k * 2][j]) *
               (A[i][k * 2] + B[k * 2 + 1][j]);
        }
    }
}
// IV
if (N % 2) {
    for (int i = 0; i < M; i++){
        for (int j = 0; j < Q; j++){
            C[i][j] = C[i][j] + A[i][N - 1] * B[N - 1][j];
        }
    }
}
```

Общая трудоемкость первого этапа I:

$$f_I = \frac{19}{2} \cdot M \cdot N + 6 \cdot M + 2 \tag{2.3}$$

Общая трудоемкость второго этапа II:

$$f_{II} = \frac{19}{2} \cdot Q \cdot N + 6 \cdot Q + 2 \tag{2.4}$$

Общая трудоемкость третьего этапа III:

$$f_{III} = 16 \cdot M \cdot N \cdot Q + 13 \cdot M \cdot Q + 4 \cdot M + 2 \tag{2.5}$$

Общая трудоемкость четвертого этапа:

$$f_{IV} = \begin{cases} 16 \cdot M \cdot Q + 4 \cdot M + 3, & \text{худший случай при четном N} \\ 1, & \text{лучший случай при нечетном N} \end{cases} \tag{2.6}$$

Итоговая трудоемкость алгоритма в лучшем случае:

$$f_{win} = 16 \cdot M \cdot N \cdot Q + \frac{19 \cdot M \cdot N}{2} + \frac{19 \cdot Q \cdot N}{2} + 13 \cdot M \cdot Q + 4 \cdot M + 6 \cdot Q + 6 \cdot M + 7 \tag{2.7}$$

Итоговая трудоемкость алгоритма в худшем случае:

$$f_{win} = 16 \cdot M \cdot N \cdot Q + \frac{19 \cdot M \cdot N}{2} + \frac{19 \cdot Q \cdot N}{2} + 29 \cdot M \cdot Q + 8 \cdot M + 6 \cdot Q + 6 \cdot M + 10 \tag{2.8}$$

2.2.3 Оптимизированный алгоритм Винограда

По варианту алгоритм был оптимизирован: ввод инкремента +=, использование двоичного сдвига влево вместо умножения на 2, введение декремента при вычислении вспомогательных массивов.

Листинг 3 – Оптимизированный алгоритм умножения матриц Винограда

```
// I - filling of MulH
for (int i = 0; i < M; i++){
    for (int k = 0; k < N / 2; k++){
        MulH[i] -= A[i][k << 1] * A[i][k << 1 + 1];
    }
}
// II - filling of MulV
for (int i = 0; i < Q; i++){
    for (int k = 0; k < N / 2; k++){
        MulV[i] -= B[k << 1][i] * B[k << 1 + 1][i];
    }
}
// III - filling of C
for (int i = 0; i < M; i++) {
    for (int j = 0; j < Q; j++){
        C[i][j] = MulH[i] + MulV[j];
        for (int k = 0; k < N / 2; k++){
            C[i][j] += (A[i][k << 1 + 1] + B[k << 1][j]) * (A[i][k
               << 1] + B[k << 1 + 1][j]);
        }
    }
}
// IV
if (N % 2) {
    for (int i = 0; i < M; i++){
        for (int j = 0; j < Q; j++){
            C[i][j] += A[i][N - 1] * B[N - 1][j];
        }
    }
}
```

Общая трудоемкость первого этапа I:

$$f_I = \frac{15}{2} \cdot M \cdot N + 6 \cdot M + 2 \tag{2.9}$$

Общая трудоемкость второго этапа II:

$$f_{II} = \frac{15}{2} \cdot Q \cdot N + 6 \cdot Q + 2 \tag{2.10}$$

Общая трудоемкость третьего этапа III:

$$f_{III} = \frac{27}{2} \cdot M \cdot N \cdot Q + 12 \cdot M \cdot Q + 4 \cdot M + 2 \tag{2.11}$$

Общая трудоемкость четвертого этапа:

$$f_{IV} = \begin{cases} 15 \cdot M \cdot Q + 4 \cdot M + 3, & \text{худший случай при четном N} \\ 1, & \text{лучший случай при нечетном N} \end{cases} \tag{2.12}$$

Итоговая трудоемкость алгоритма в лучшем случае:

$$f_{win} = \frac{27 \cdot M \cdot N \cdot Q}{2} + \frac{15 \cdot M \cdot N}{2} + \frac{15 \cdot Q \cdot N}{2} + 12 \cdot M \cdot Q + 4 \cdot M + 6 \cdot Q + 6 \cdot M + 7 \tag{2.13}$$

Итоговая трудоемкость алгоритма в худшем случае:

$$f_{win} = \frac{27 \cdot M \cdot N \cdot Q}{2} + \frac{15 \cdot M \cdot N}{2} + \frac{15 \cdot Q \cdot N}{2} + 12 \cdot M \cdot Q + 19 \cdot M + 6 \cdot Q + 10 \cdot M + 10 \tag{2.14}$$

2.3 Разработка алгоритмов

На вход каждому алгоритму подается две матрицы и их размеры — количество строк и столбцов.

На рисунках 2.1 — 2.3 приведены стандартный алгоритм и алгоритм Винограда умножения матриц.

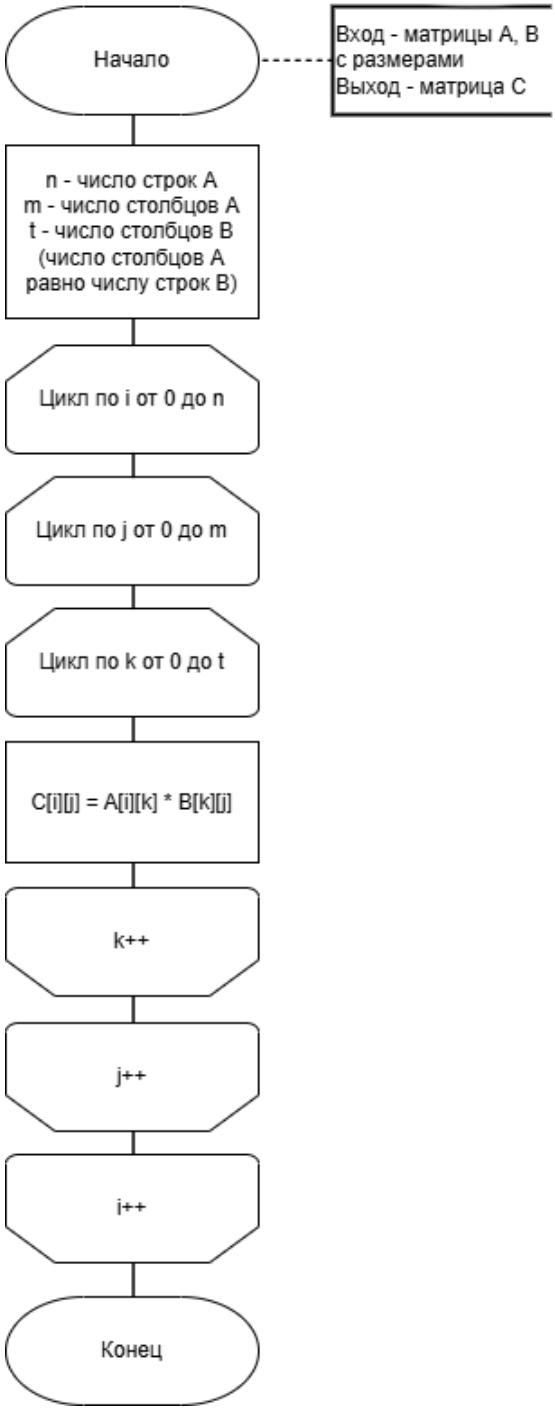

Рисунок 2.1 – Стандартный алгоритм умножения матриц

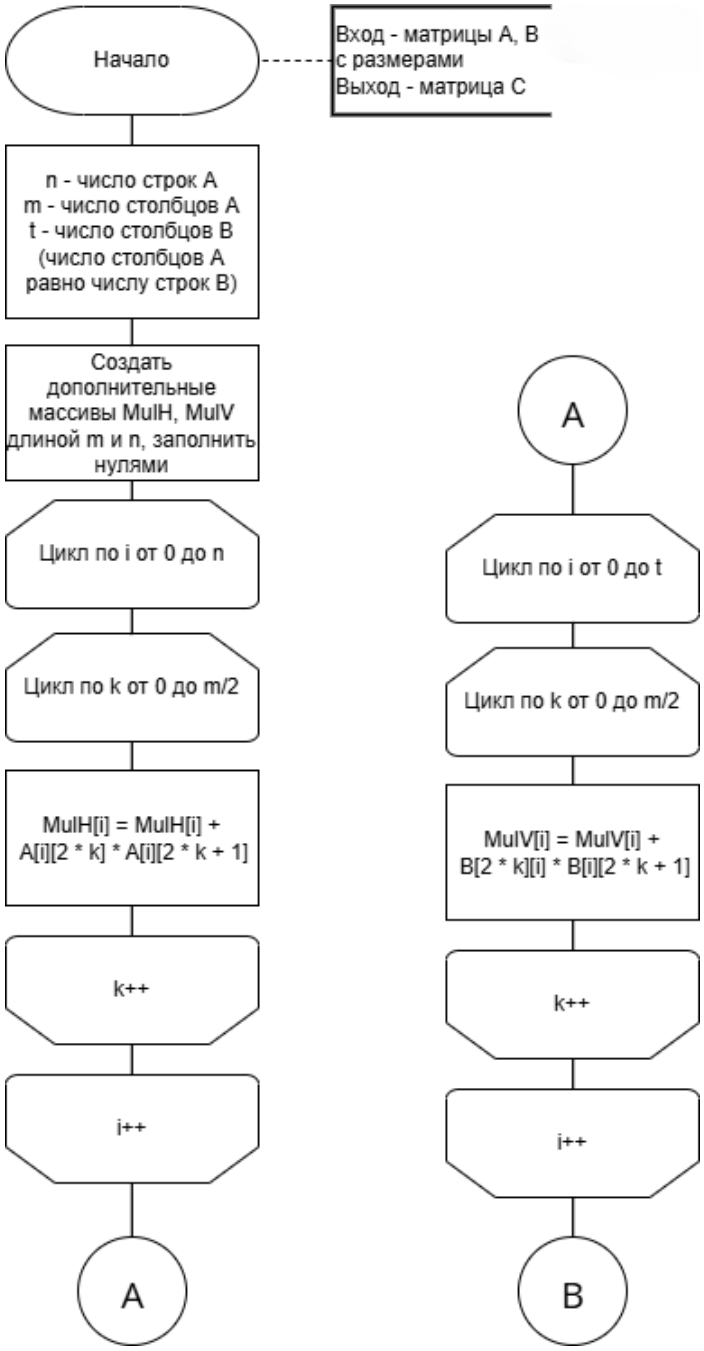

Рисунок 2.2 – Алгоритм Винограда часть 1

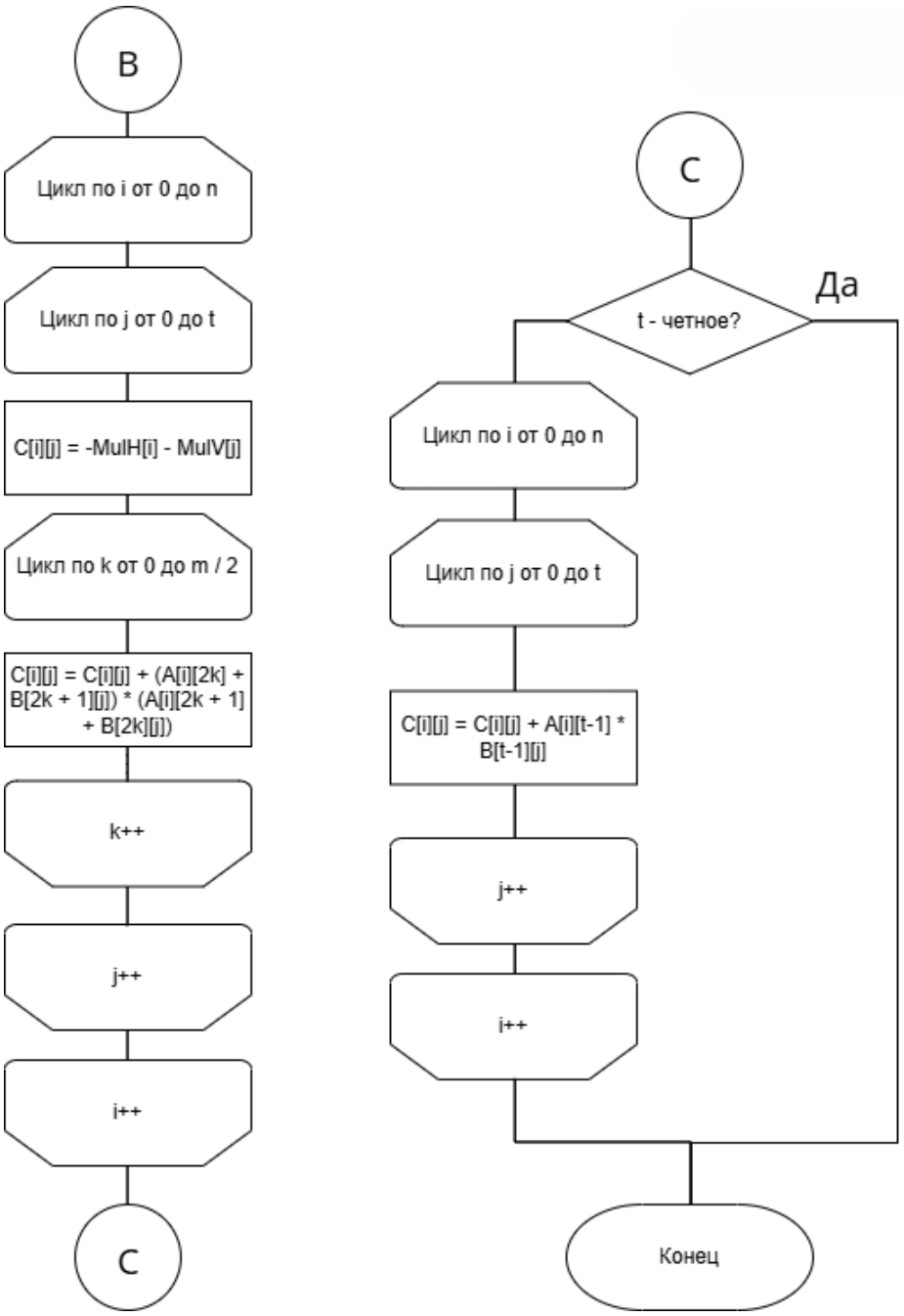

Рисунок 2.3 – Алгоритм Винограда часть 2

Вывод

В данном разделе были разработаны алгоритмы умножения матриц.

3 Технологическая часть

В данном разделе будут приведены данные о выбранном языке программирования, коды алгоритмов и тесты для каждого алгоритма.

3.1 Требования к ПО

Реализуемое ПО будет давать возможность работы в двух режимах: умножение двух введенных пользователем матриц, а также массовый замер времени умножения случайно сгенерированных квадратных матриц указанного размера.

3.2 Язык программирования

В данной работе для реализации алгоритмов был выбран язык программирования *C* [3]. Язык предоставляет возможность работы с матрицами различных размеров, а также возможность создания пользовательского интерфейса. Также язык позволяет производить замеры процессорного времени выполнения функций с использованием функции clock [2].

3.3 Реализация алгоритмов

В листингах 1 — 3 представлены реализации алгоритма поиска полным перебором и алгоритма бинарного поиска.

Листинг 1 – Реализация стандартного алгоритма умножения матриц

```
int** matrix_mul(int** fir, int c_row1, int c_col1, int** sec, int
   c_row2, int c_col2, double *process_time)
{
    int c_row = c_row1;
    int c_col = c_col2;
    if (c_col1 != c_row2)
        return NULL;
    int** result = allocate_matrix(c_row, c_col);
    if (!result)
        return NULL;
    clock_t begin = clock();
    for (int i = 0; i < c_row; i++)
        for (int j = 0; j < c_col; j++){
            result[i][j] = 0;
            for (int k = 0; k < c_col1; k++)
                result[i][j] = result[i][j] + fir[i][k] * sec[k][j];
        }
    *process_time = (double) (clock() - begin) / CLOCKS_PER_SEC;
    return result;
}
```

Листинг 2 – Реализация алгоритма Винограда

```
int** matrix_mul_winograd(int** fir, int c_row1, int c_col1, int**
   sec, int c_row2, int c_col2, double *process_time)
{
    int c_row = c_row1;
    int c_col = c_col2;
    int* mulH = malloc(c_row1 * sizeof(int));
    if (!mulH)
        return NULL;
    for (int i = 0; i < c_row1; i++)
        mulH[i] = 0;
    int* mulV = malloc(c_col2 * sizeof(int));
    if (!mulV){
        free(mulH);
        return NULL;
    }
    for (int i = 0; i < c_col2; i++)
        mulV[i] = 0;
    int** result = allocate_matrix(c_row, c_col);
```

```
    if (!result) {
        free(mulH);
        free(mulV);
        return NULL;
    }
    clock_t begin = clock();
    for (int i = 0; i < c_row1; i++){
        for (int j = 0; j < c_col1 / 2; j++){
            mulH[i] = mulH[i] + fir[i][j * 2] * fir[i][j * 2 + 1];
        }
    }
    for (int i = 0; i < c_col2; i++){
        for (int j = 0; j < c_row2 / 2; j++){
            mulV[i] = mulV[i] + sec[j * 2][i] * sec[j * 2 + 1][i];
        }
    }
    for (int i = 0; i < c_row; i++) {
        for (int j = 0; j < c_col; j++){
            result[i][j] = -(mulH[i] + mulV[j]);
            for (int k = 0; k < c_col1 / 2; k++){
                result[i][j] = result[i][j] + (fir[i][k * 2 + 1]+
                   sec[k * 2][j]) * (fir[i][k * 2] + sec[k * 2 +
                   1][j]);
            }
        }
    }
    if (c_col1 % 2) {
        for (int i = 0; i < c_row; i++){
            for (int j = 0; j < c_col; j++){
                result[i][j] = result[i][j] + fir[i][c_col1 - 1] *
                   sec[c_col1 - 1][j];
            }
        }
    }
    *process_time = (double) (clock() - begin) / CLOCKS_PER_SEC;
    free(mulH);
    free(mulV);

    return result;
}
```

Листинг 3 – Реализация оптимизированного алгоритма винограда

```
int** matrix_mul_winograd_optimized(int** fir, int c_row1, int
   c_col1, int** sec, int c_row2, int c_col2, double *process_time)
{
    int c_row = c_row1;
    int c_col = c_col2;
    int* mulH = malloc(c_row1 * sizeof(int));
    if (!mulH)
        return NULL;
    for (int i = 0; i < c_row1; i++)
        mulH[i] = 0;
    int* mulV = malloc(c_col2 * sizeof(int));
    if (!mulV){
        free(mulH);
        return NULL;
    }
    for (int i = 0; i < c_col2; i++)
        mulV[i] = 0;
    int** result = allocate_matrix(c_row, c_col);
    if (!result) {
        free(mulH);
        free(mulV);
        return NULL;
    }
    clock_t begin = clock();
    for (int i = 0; i < c_row1; i++){
        for (int j = 0; j < c_col1 / 2; j++){
            mulH[i] -= fir[i][j << 1] * fir[i][(j << 1) + 1];
        }
    }
    for (int i = 0; i < c_col2; i++){
        for (int j = 0; j < c_row2 / 2; j++){
            mulV[i] -= sec[j << 1][i] * sec[(j << 1) + 1][i];
        }
    }
    for (int i = 0; i < c_row; i++) {
        for (int j = 0; j < c_col; j++){
            result[i][j] = mulH[i] + mulV[j];
            for (int k = 0; k < c_col1 / 2; k++){
                result[i][j] += (fir[i][(k << 1) + 1]+ sec[k <<
                   1][j]) * (fir[i][k << 1] + sec[(k << 1) + 1][j]);
```

```
            }
        }
    }
    if (c_col1 % 2) {
        for (int i = 0; i < c_row; i++){
            for (int j = 0; j < c_col; j++){
                result[i][j] += fir[i][c_col1 - 1] * sec[c_col1 -
                    1][j];
            }
        }
    }
    *process_time = (double) (clock() - begin) / CLOCKS_PER_SEC;
    free(mulH);
    free(mulV);

    return result;
}
```

3.4 Функциональные тесты

В листинге 4 описан вывод программы для различных функциональных тестов. Все тесты программа прошла успешно.

Листинг 4 – Функциональные тесты

```
# Test 1
Please enter the number of rows and columns:
3 4
Enter the elements of the matrix:
1 2 3 4 5 6 7 8 9 10 11 12
Please enter the number of rows and columns:
4 2
Enter the elements of the matrix:
1 2 3 4 5 6 7 8
Result (default algorithm):
50 60
114 140
178 220
Result (Winograd algorithm):
50 60
114 140
178 220
Result (Optimized Winograd algorithm):
50 60
114 140
178 220

# Test 2
Please enter the number of rows and columns:
1 5
Enter the elements of the matrix:
1 2 3 4 5
Please enter the number of rows and columns:
5 1
Enter the elements of the matrix:
1 2 3 4 5
Result (default algorithm):
55
Result (Winograd algorithm):
55
Result (Optimized Winograd algorithm):
```

```
55

# Test 3
Please enter the number of rows and columns:
1 5
Enter the elements of the matrix:
1 2 3 4 5
Please enter the number of rows and columns:
5 3
Enter the elements of the matrix:
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
Result (default algorithm):
135 150 165
Result (Winograd algorithm):
135 150 165
Result (Optimized Winograd algorithm):
135 150 165
```

Вывод

В данном разделе были приведены коды реализаций алгоритмов умножения матриц, а также функциональные тесты.

4 Исследовательская часть

В данном разделе будут представлены результаты исследования эффективности работы алгоритмов по времени. Замеры проводились на плате STM32H735IGK6 с 564КБ оперативной памяти и 1МБ флэш-памяти [4].

4.1 Замеры времени работы алгоритмов

Результаты замеров времени перемножения матриц от их размеров представлены на графиках 4.1 — 4.2. Замеры производились на квадратных матрицах размерами от 30 на 30 до 100 на 100.

Таблица 4.1 – Результаты замеров времени для четных размеров матриц (ХС), с

Размер	Стандартный	Виноград	Виноград (опт)
30	0.002	0.002	0.001
40	0.004	0.003	0.003
50	0.007	0.005	0.006
60	0.019	0.014	0.012
70	0.022	0.017	0.013
80	0.031	0.025	0.023
90	0.043	0.036	0.032
100	0.062	0.046	0.041

Таблица 4.2 – Результаты замеров времени для нечетных размеров матриц (ЛС), с

Размер	Стандартный	Виноград	Виноград (опт)
31	0.001	0.002	0.002
41	0.004	0.004	0.004
51	0.008	0.006	0.006
61	0.014	0.013	0.012
71	0.021	0.019	0.017
81	0.032	0.028	0.025
91	0.044	0.039	0.035
101	0.063	0.052	0.047

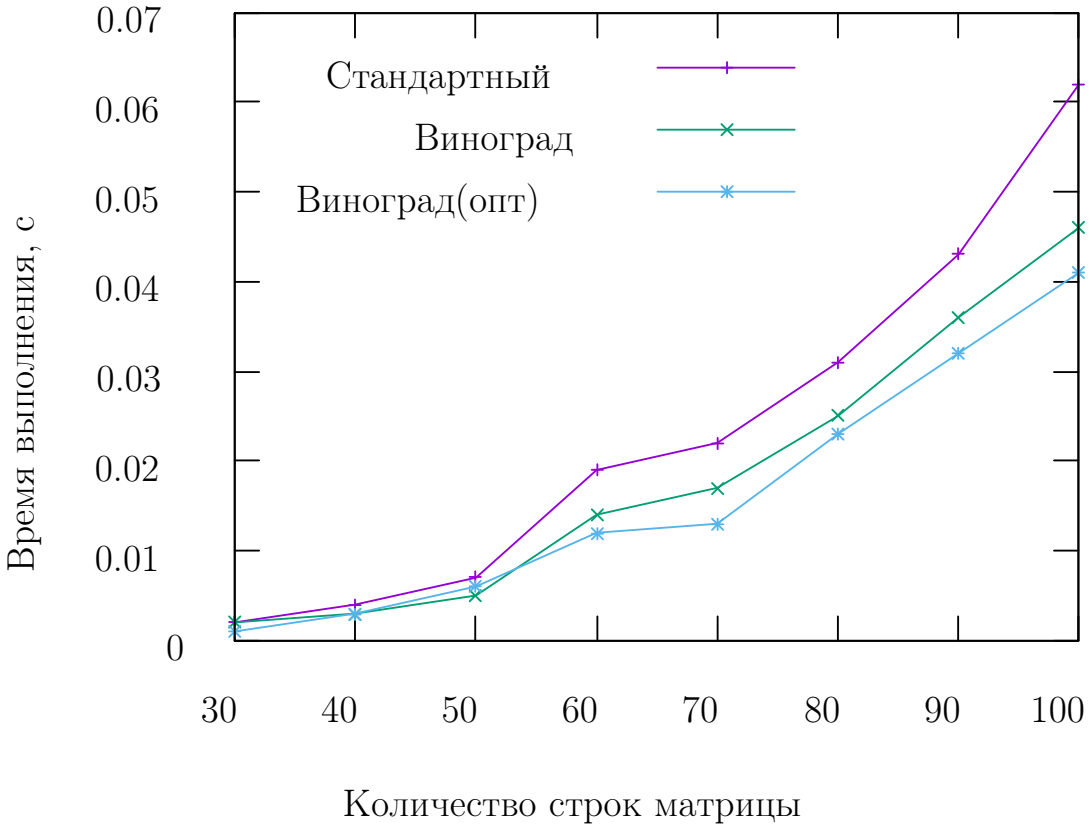

Рисунок 4.1 – Сравнение для четных размеров матриц

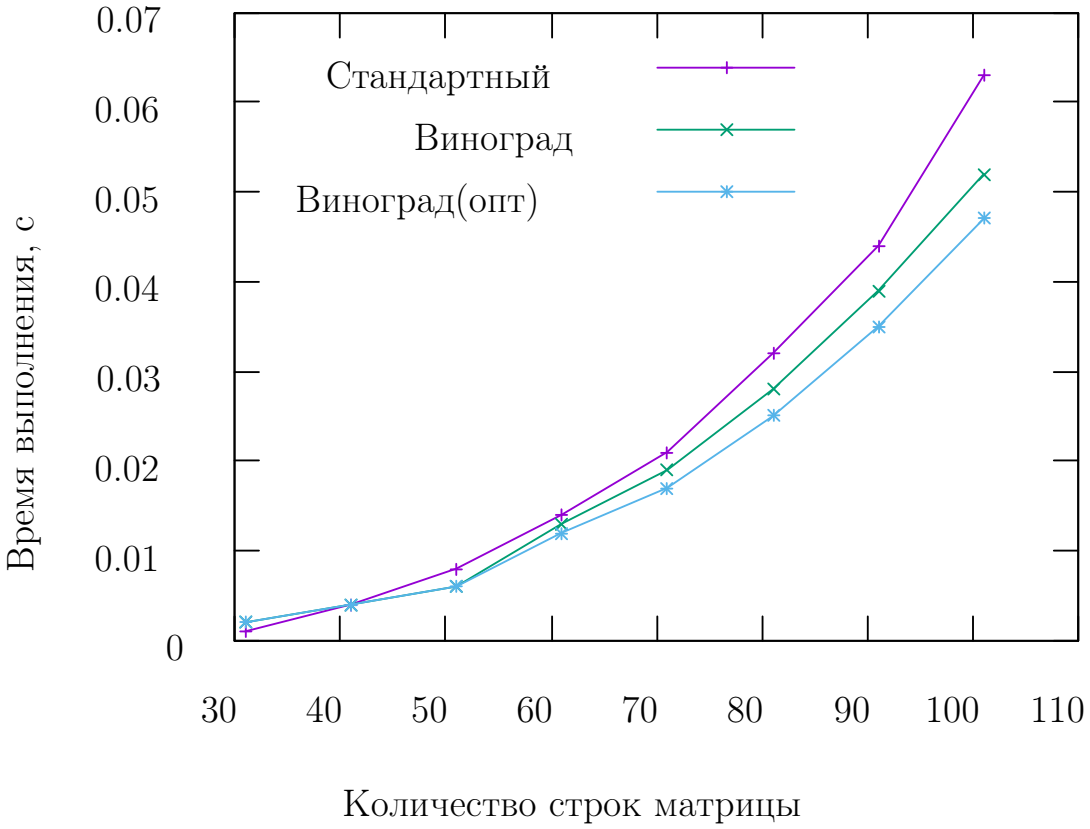

Рисунок 4.2 – Сравнение для нечетных размеров матриц

Вывод

Алгоритм Винограда является более быстрым, чем классический алгоритм умножения матриц, при дополнительных оптимизациях его можно сделать еще быстрее. Однако, алгоритм Винограда требует использования дополнительной памяти под массивы MulH и MulV. Для случая квадратных матриц вида $N \times N$ дополнительные массивы занимают долю равную $\frac{N+N}{2 \cdot N^2+2 \cdot N} = \frac{1}{N+1}$ от общей памяти, используемой для вычислений (не считая матрицы-результата).

ЗАКЛЮЧЕНИЕ

В ходе выполнения данной лабораторной работы были проанализированы классический алгоритм и алгоритм Винограда умножения матриц, а также решены следующие задачи:

— проанализирован классический алгоритм умножения матриц и алгоритм Винограда;

— реализованы указанные алгоритмы;

— проведено сравнение эффективности данных алгоритмов по времени выполнения на плате STM;

— описаны и обоснованы полученные результаты в отчете о выполненной лабораторной работе.

Алгоритм Винограда является более быстрым, чем классический алгоритм умножения матриц, при дополнительных оптимизациях его можно сделать еще быстрее. Однако, алгоритм Винограда требует использования дополнительной памяти под массивы MulH и MulV. Для случая квадратных матриц вида $N \times N$ дополнительные массивы занимают долю равную $\frac{N+N}{2 \cdot N^2+2 \cdot N} = \frac{1}{N+1}$ от общей памяти, используемой для вычислений (не считая матрицы-результата).

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Линейная алгебра, учебное пособие [Электронный ресурс]. Режим доступа: https://elar.urfu.ru/bitstream/10995/78551/1/978-5-7996-2776-8_2019.pdf?ysclid=m29hw3nfrq373542650 (дата обращения: 14.10.2024).

[2] Функция clock() [Электронный ресурс]. Режим доступа: https://en.cppreference.com/w/c/chrono/clock (дата обращения: 14.10.2024).

[3] Стандарт языка Си [Электронный ресурс]. Режим доступа: https://cs.petrsu.ru/~akolosov/inf/2009/docs/c99__iso-9899-tc2.pdf (дата обращения: 14.10.2024).

[4] Discovery kit with STM32H735IG MCU [Электронный ресурс]. Режим доступа: https://www.st.com/en/evaluation-tools/stm32h735g-dk.html (дата обращения: 14.10.2024).