Федеральное государственное бюджетное образовательное учреждение высшего образования «Московский государственный технический университет имени Н. Э. Баумана (национальный исследовательский университет»)

Курс: «Анализ алгоритмов»

Лабораторная работа №4

Тема работы: «Реализация многопоточности для алгоритма Винограда»

Студент: Аминов Т. С.
Преподаватели: Волкова Л. Л.
Строганов Ю. В.
Группа: ИУ7-55Б

2019

Содержание

Введение

Цель лабораторной работы: изучение метода динамического программирования на материале оптимизированного алгоритма Винограда и реализация для него многопоточности.

Задачи работы:

1) изучение алгоритма умножения матриц по Винограду;
2) оптимизация алгоритма Винограда;
3) реализация многопоточности для оптимизированного алгоритма Винограда;
3) применение метода динамического программирования для реализации указанных алгоритмов;
4) приобретение практических навыков реализации указанных алгоритмов: оптимизированного алгоритма Винограда, разбитого на потоки;
5) сравнительный анализ по затрачиваемым ресурсам (времени) при разном количестве рабочих потоков;
7) описание и обоснование полученных результатов в отчете о выполненной лабораторной работе, выполненного как расчётно-пояснительная записка к работе.

1 Аналитический раздел

В данном разделе анализируются алгоритмы вычисления произведения матриц по стандартному алгоритму и по алгоритму Винограда.

1.1 Описание алгоритмов

Матрица A размера $m \times n$ — это прямоугольная таблица чисел, расположенных в m строках и n столбцах:

$$A = \begin{pmatrix} a_{11} & a_{12} & ... & a_{1n} \\ a_{21} & a_{22} & ... & a_{2n} \\ ... & ... & ... & ... \\ a_{mn} & a_{m2} & ... & a_{mn} \end{pmatrix}$$

где $a_{ij}(i = 1, \ldots, m; j = 1, \ldots, n)$ — это элементы матрицы A. Первый индекс i — это номер строки, второй индекс j — это номер столбца, на пересечении которых расположен элемент a_{ij} [2].

Матрицы широко применяются в математике для компактной записи систем линейных алгебраических или дифференциальных уравнений.

1.1.1 Стандартный алгоритм

Для вычисления произведения двух матриц каждая строка первой почленно умножается на каждый столбец второй. Затем подсчитывается сумма таких произведений и записывается в соответствующую клетку результата [1]:

$$\begin{bmatrix} a_{11} & a_{12} & ... & a_{1q} \\ a_{21} & a_{22} & ... & a_{2q} \\ ... & ... & ... & ... \\ a_{m1} & a_{m2} & ... & a_{mq} \end{bmatrix} \times \begin{bmatrix} b_{11} & b_{12} & ... & b_{1n} \\ b_{21} & b_{22} & ... & b_{2n} \\ ... & ... & ... & ... \\ b_{q1} & b_{q2} & ... & b_{qn} \end{bmatrix} =$$

$$= \begin{bmatrix} a_{11} * b_{11} + ... + a_{1q} * b_{q1} & ... & ... & a_{11} * b_{1n} + ... + a_{1q} * b_{qn} \\ a_{21} * b_{11} + ... + a_{2q} * b_{q1} & ... & ... & a_{21} * b_{1n} + ... + a_{2q} * b_{qn} \\ ... & ... & ... & ... \\ a_{m1} * b_{11} + ... + a_{mq} * b_{q1} & ... & ... & a_{m1} * b_{1n} + ... + a_{mq} * b_{qn} \end{bmatrix}.$$

1.1.2 Алгоритм 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 $V = (v_1, v_2, v_3, v_4)$ и $W = (w_1, w_2, w_3, w_4)$. Их скалярное произведение равно:

$$V \cdot W = v_1w_1 + v_2w_2 + v_3w_3 + v_4w_4.$$

Это равенство можно представить так:

$$V \cdot W = (v_1 + w_2) * (w_1 + v_2) + (v_3 + w_4) * (w_3 + v_4) - v_1v_2 - v_3v_4 - w_1w_2 - w_3w_4.$$

Такой подход позволяет вычислять $-v_1v_2 - v_3v_4$ и $-w_1w_2 - w_3w_4$ заранее и запомнить для каждой строки первой матрицы и для каждого столбца второй. Также в алгоритме Винограда содержится меньше затратных по времени операций умножения, по сравнению со стандартным[1].

Вывод

В данном разделе были рассмотрены алгоритмы стандартного умножения матриц и алгоритм Винограда, который решает ту же задачу за меньшее количество операций путем предварительных вычислений частей произведения и путем использования меньшего количества операции умножения.

2 Конструкторский раздел

В данном разделе описываются шаги по оптимизации алгоритма Винограда и адаптация его под многопоточное выполнение, содержатся схемы стандартного алгоритма умножения матриц и алгоритма Винограда в оптимизированной и неоптимизированной реализациях.

2.1 Разработка алгоритмов

2.1.1 Неоптимизированнный алгоритм Винограда

На рис. 1 и 2 приведена схема неоптимизированного алгоритма Винограда.

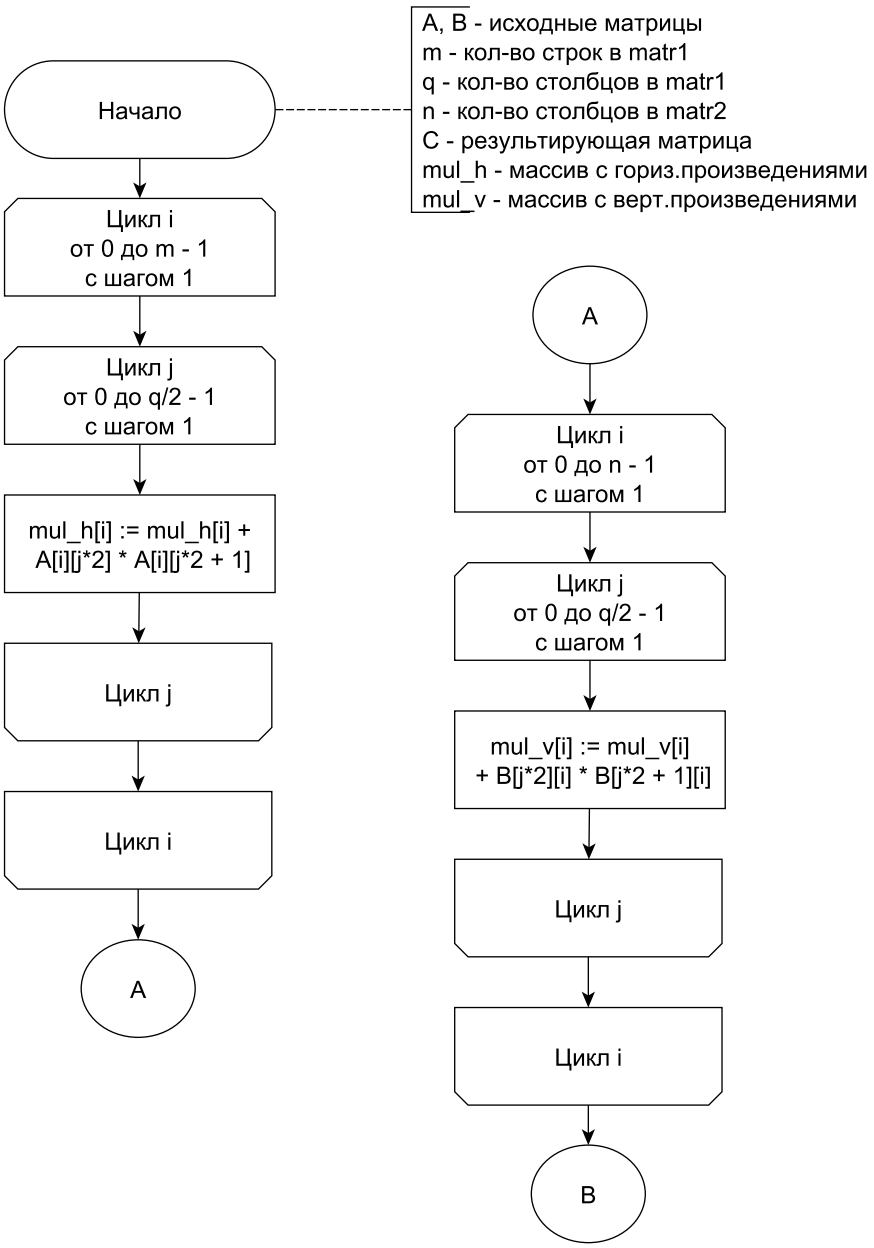

Рис. 1: Схема алгоритма умножения матриц по Винограду (часть 1)

2.1.2 Оптимизация алгоритма Винограда

Заполнение массива под горизонтальные (вертикальные) произведения:

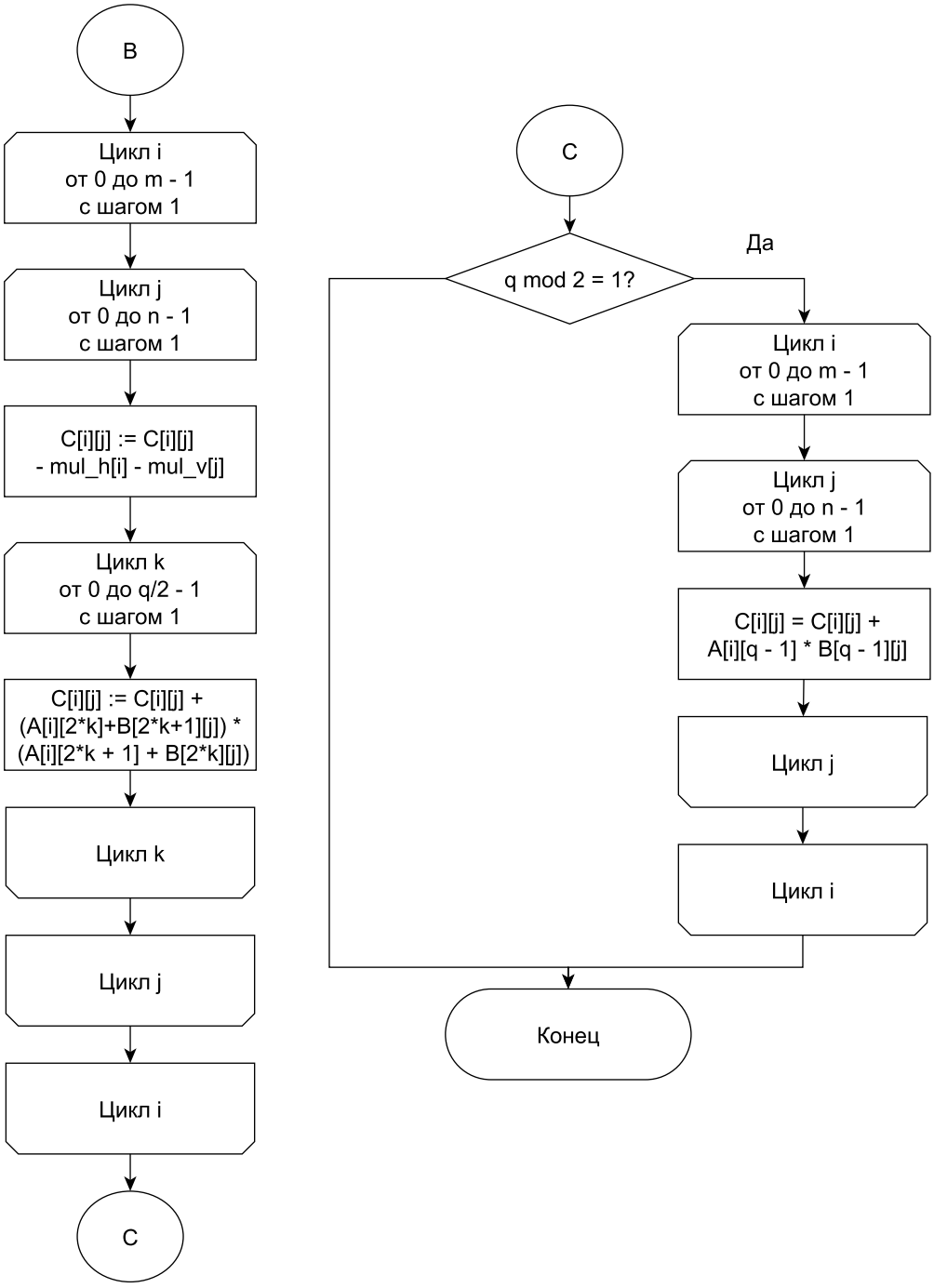

Рис. 2: Схема алгоритма умножения матриц по Винограду (часть 2)

1) замена = на +=;
2) замена во внутреннем цикле шага цикла с 1 на 2 ⇒ происходит замена j*2 на j.

Тройной цикл:

1) замена = на +=;
2) замена в цикле по q шага цикла с 1 на 2 ⇒ происходит замена k*2 на k;
3) вычисление $q2 = q - 1$ заранее;
4) использование буфера для накопления результата по циклу q и занесение результата в $res\_matr[i][j]$ после цикла;

5) вычисление горизонтального и вертикального произведения заранее отрицательным.

Условный переход:

1) замена = на +=;
2) замена во внутреннем цикле шага цикла с 1 на 2 $\Rightarrow$ происходит замена j*2 на j;
3) вычисление q2 = q - 1 заранее.

2.1.3 Реализация многопоточности

Пусть разрешено использовать до N рабочих потоков, тогда распараллелить алгоритм Винограда можно следующими способами.

1) Выполнять вычисления горизонтальных и вертикальных произведений в двух рабочих потоках.
2) В тройном цикле организовать внешний цикл так, чтобы первый рабочий поток вычислял элементы первой строки результирующей матрицы, затем элементы (1 + N)-ой строки, элементы (1 + 2*N)-ой строки и т.д., второй рабочий поток аналогично вычисляет строки 2, 2 + N, 2 + 2*N, ... Таким образом i-й рабочий поток вычисляет строки i, i + N, i + 2*N, ... , пока i $<$ m, где m - длина первой матрицы.
3) Аналогично 2) организовать внешний цикл в условном переходе.

Таким образом, сначала происходит параллельный расчёт горизонтальных и вертикальных произведений в двух рабочих потоках, затем каждая строка результирующей матрицы вычисляется на отдельном рабочем потоке и затем, если матрица нечетной размерности, дополнительные вычисления (в условном переходе) аналогично выполняются для каждой строки результирующей матрицы на отдельном рабочем потоке.

На рис. 3, 4, 5, 6 и 7 представлены схемы оптимизированного алгоритма Винограда, разбитого на потоки.

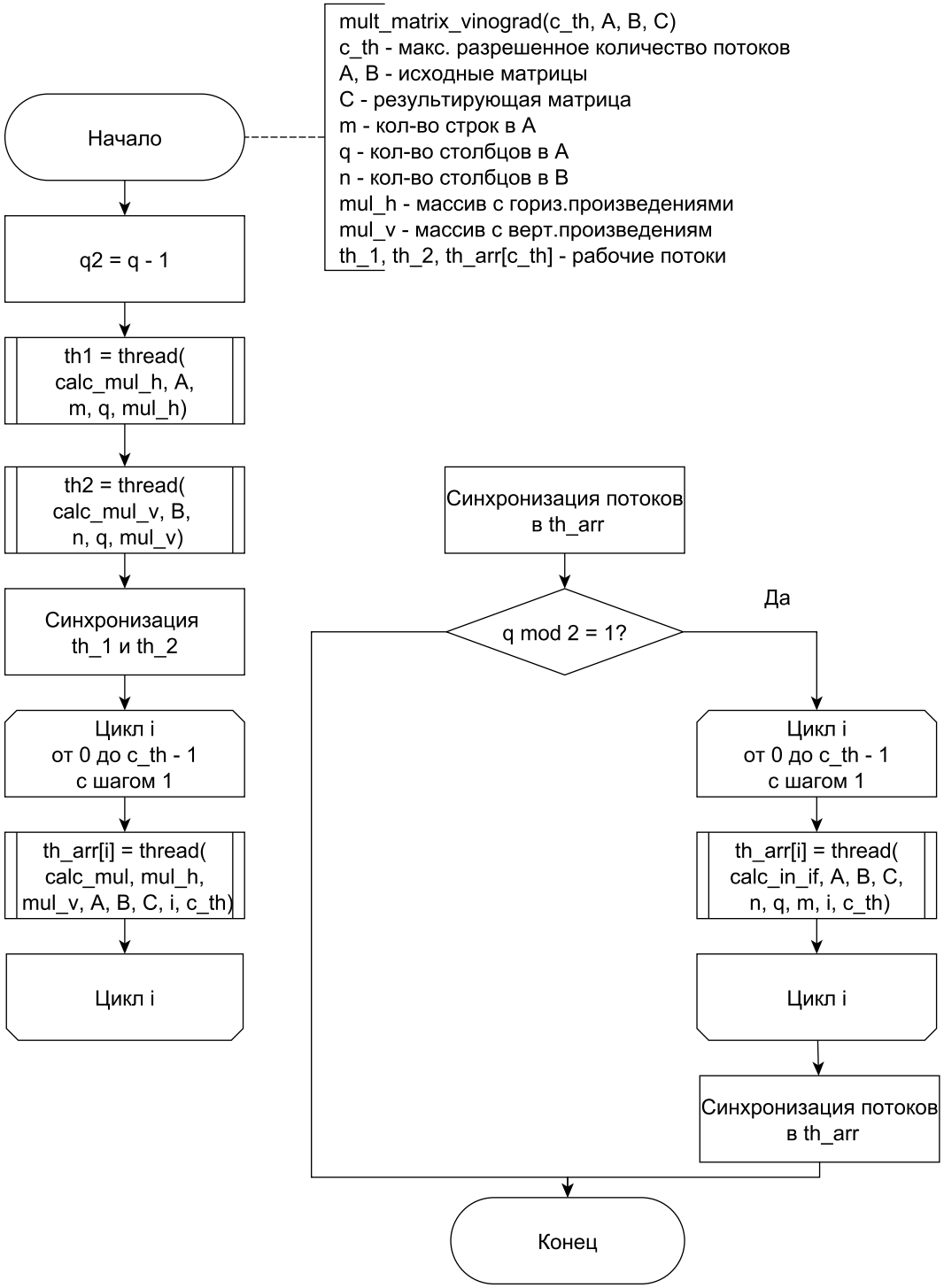

Рис. 3: Схема основной функции

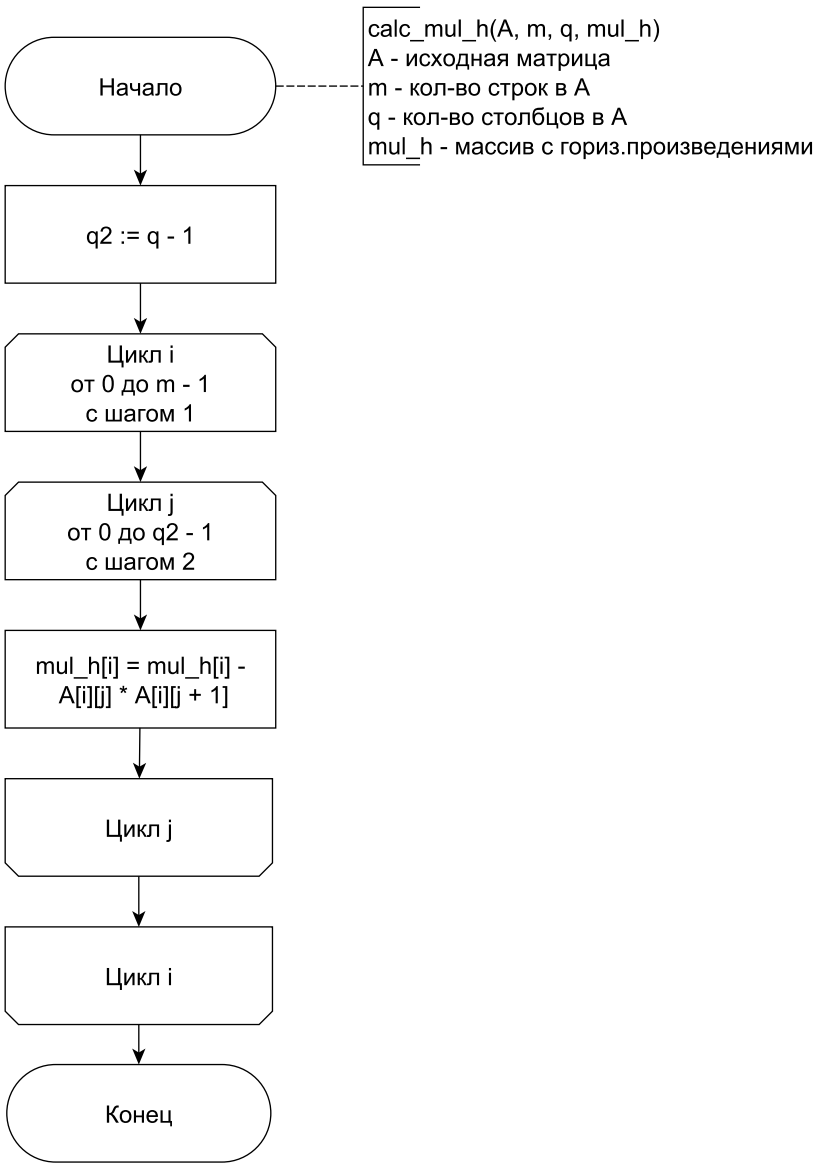

Рис. 4: Схема функции, вычисляющей горизонтальные произведений

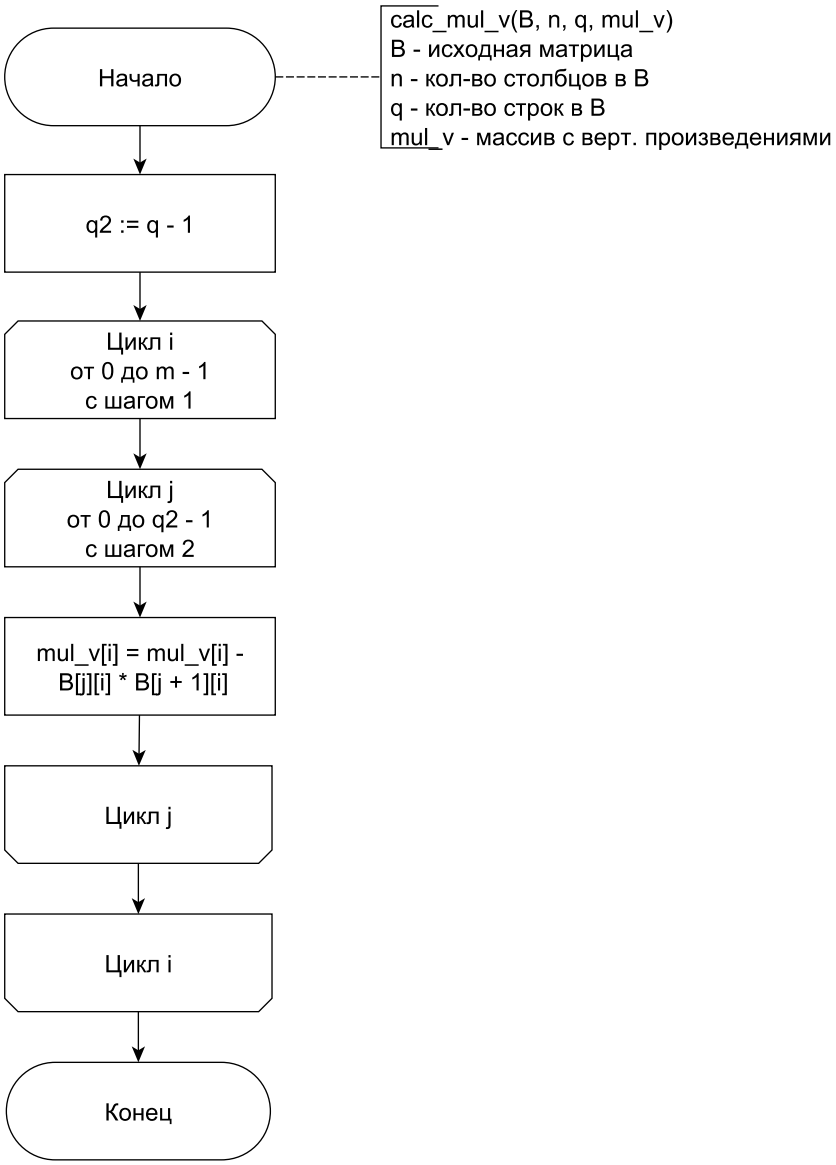

Рис. 5: Схема функции, вычисляющей горизонтальные произведений

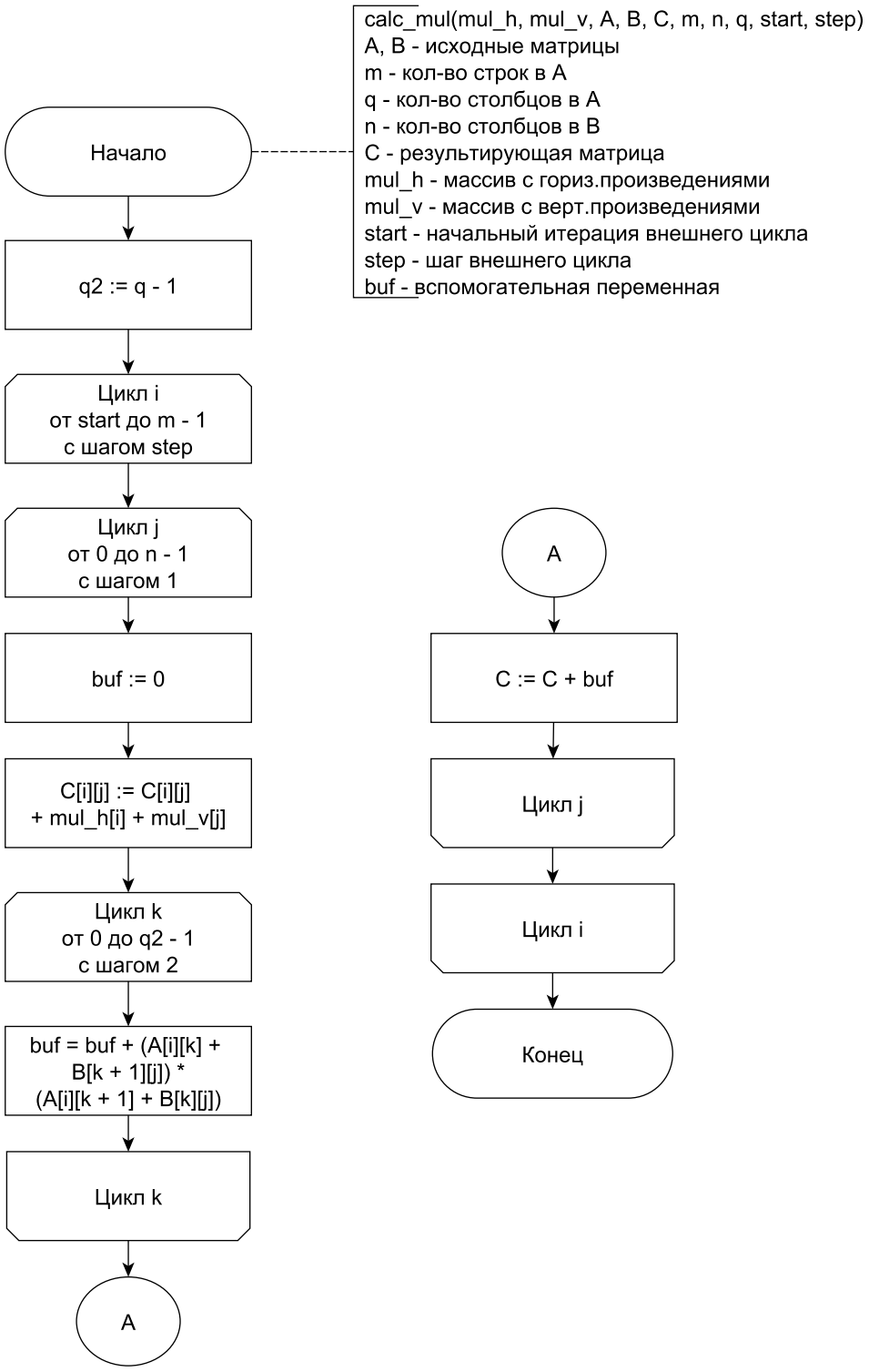

Рис. 6: Схема функции вычислений в тройном цикле

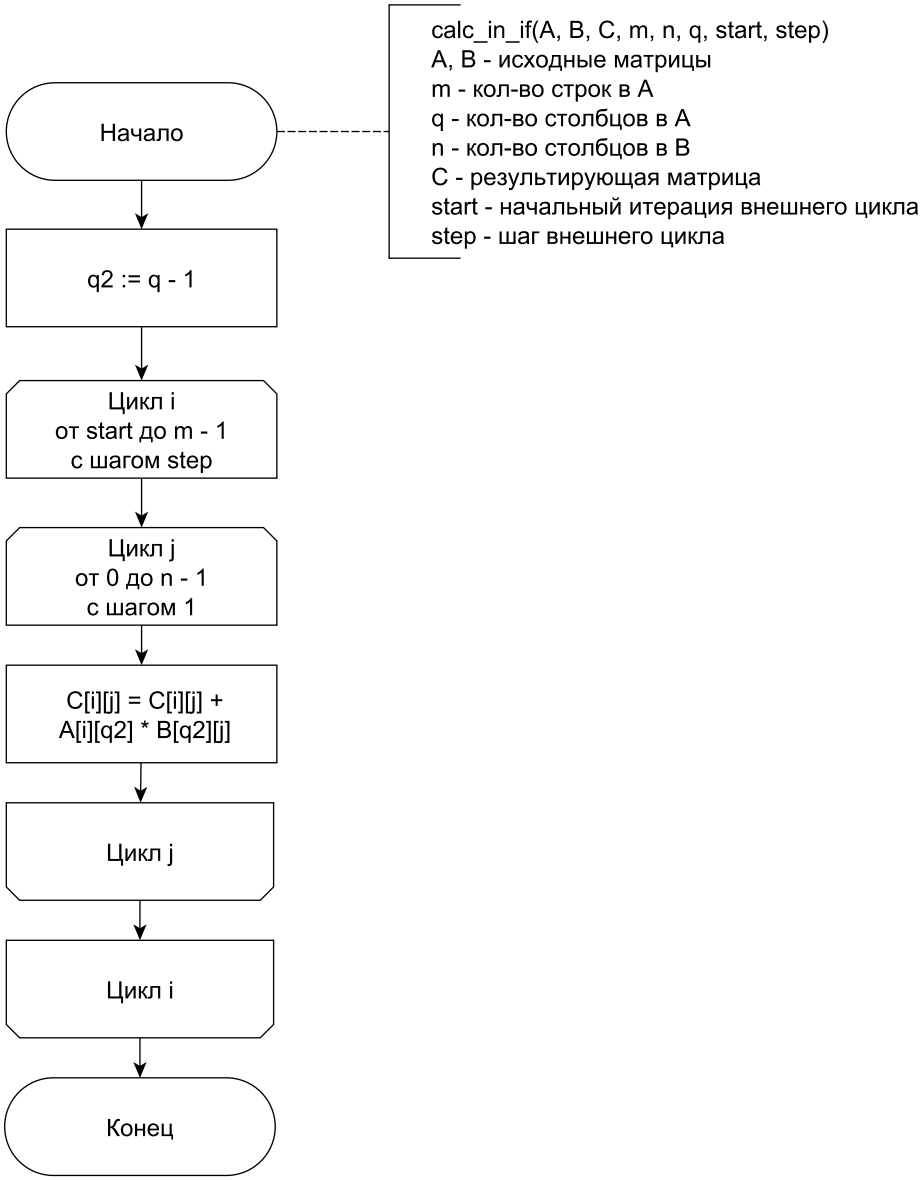

Рис. 7: Схема функции вычислений внутри условного перехода

Вывод

В данном разделе были описаны шаги по оптимизации алгоритма Винограда, была предложена модификация алгоритма для многопоточного выполнения и были представлены схемы оптимизированного алгоритма Винограда и оптимизированного алгоритма Винограда, адаптированного под выполнение на потоках.

3 Технологический раздел

В данном разделе будут описаны требования к программному обеспечению и средства реализации, приведены листинг программ и тестовые данные.

3.1 Требования к программному обеспечению

Входные данные:

1) три целых положительных числа - размерности матриц: M, N и Q.
2) две матрицы размера M x Q и Q x N, заполненные целыми числами
3) целое число - максимальное количество рабочих потоков

Выходные данные: матрица размера M x N, полученная в результате умножения исходных, с помощью оптимизированного алгоритма Винограда, разбитого на потоки.

На рис. 8 приведена функциональная схема вычисления произведения матриц по оптимизированному алгоритму Винограда с разбиением на потоки.

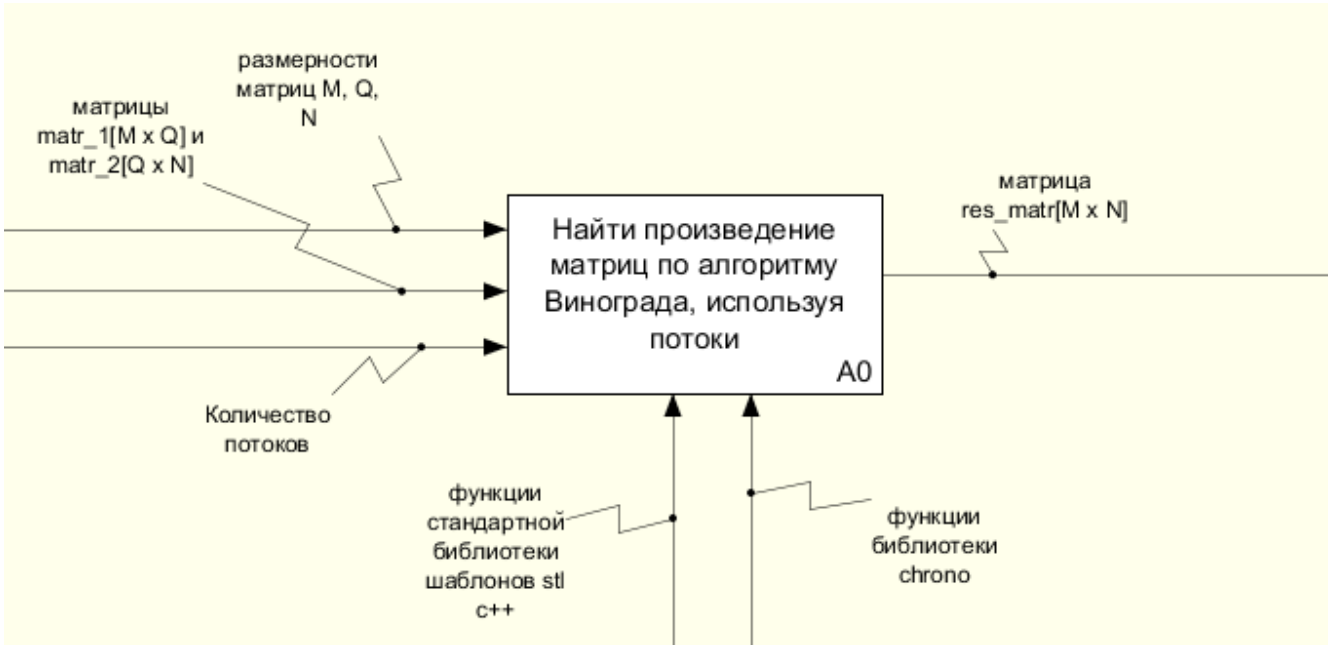

Рис. 8: Функциональная схема вычисления произведения матриц по оптимизированному алгоритму Винограда с разбиением на потоки

3.2 Средства реализации

Программа написана на языке C++, т. к. этот язык предоставляет программисту широкие возможности реализации самых разнообразных алгоритмов, обладает высокой эффективностью и значительным набором стандартных классов и процедур. В качестве среды разработки использовался фреймворк QT 5.13.1.

Для обработки матриц был использован стандартный контейнерный класс std::vector.

Для замера времени выполнения программы использовалась библиотека chrono.

3.3 Листинг программы

В листинге 1 содержится основная функция умножения матриц по оптимизировнаному алгоритму Винограду с разделением на потоки. В листингах 2, 3, 4 и 5 представлены функции, которые обрабатываются на рабочих потоках.

Листинг 1: Основная функция

```
  void mult_matrix_vinograd_optimiz(int count_th, Matrix matr_1, Matrix matr_2,
                                    Matrix &res_matr) {
    size_t m = matr_1.size();
    size_t q = matr_1[0].size();
    size_t q2 = q - 1;
    size_t n = matr_2[0].size();

    Vector mul_h(m, 0);
    Vector mul_v(n, 0);

    std::thread thread_1(calc_mul_h, std::ref(matr_1), m, q2,
                         std::ref(mul_h));
    std::thread thread_2(calc_mul_v, std::ref(matr_2), n, q2,
                         std::ref(mul_v));
    thread_1.join();
    thread_2.join();

    std::vector<std::thread> thread_arr(count_th);

    for(int i = 0; i < count_th; i++) {
        thread_arr[i] = std::thread(calc_mult, std::ref(mul_h),
                                    std::ref(mul_v), std::ref(matr_1),
                                    std::ref(matr_2),
                                    std::ref(res_matr), i, count_th);
    }
    for(int i = 0; i < count_th; i++) {
        thread_arr[i].join();
    }

    if (q % 2 == 1) {
        for(int i = 0; i < count_th; i++) {
            thread_arr[i] = std::thread(calc_in_if, std::ref(matr_1), std::ref(matr_2),
                std::ref(res_matr), i, count_th);
        }
        for(int i = 0; i < count_th; i++) {
            thread_arr[i].join();
        }
    }
}
```

Листинг 2: Функция вычисления горизонтальных произведений

```
using Matrix = std::vector<std::vector<int>>;
using Vector = std::vector<int>;

void calc_mul_h(Matrix &matr_1, size_t m, size_t q2, Vector &mul_h) {
    for(size_t i = 0; i < m; i++) {
        for (size_t j = 0; j < q2; j+=2) {
            mul_h[i] -= matr_1[i][j] * matr_1[i][j + 1];
        }
    }
}
```

Листинг 3: Функция вычисления вертикальных произведений

```
void calc_mul_v(Matrix &matr_2, size_t n, size_t q2, Vector &mul_v) {
    for(size_t i = 0; i < n; i++) {
        for (size_t j = 0; j < q2; j+=2) {
```

```
            mul_v[i] -= matr_2[j][i] * matr_2[j + 1][i];
        }
    }
}
```

Листинг 4: Функция вычислений в тройном цикле

```
void calc_mult(Vector &mul_h, Vector &mul_v, Matrix &matr_1, Matrix &matr_2,
               Matrix &res_matr, size_t start, size_t step) {
    size_t m = matr_1.size();
    size_t q2 = matr_1[0].size() - 1;
    size_t n = matr_2[0].size();
    for(size_t i = start; i < m; i += step) {
        for (size_t j = 0; j < n; j++) {
            res_matr[i][j] = mul_h[i] + mul_v[j];
            int buf = 0;
            for(size_t k = 0; k < q2; k+=2) {
                buf += (matr_1[i][k] + matr_2[k + 1][j]) *
                                    (matr_1[i][k + 1] + matr_2[k][j]);
            }
            res_matr[i][j] += buf;
        }
    }
}
```

Листинг 5: Функция дополнительных вычислений для матриц нечетной размерности

```
  void calc_in_if(Matrix &matr_1, Matrix &matr_2, Matrix &res_matr,
       size_t start, size_t step) {
    size_t m = matr_1.size();
    size_t q2 = matr_1[0].size() - 1;
    size_t n = matr_2[0].size();
    for(size_t i = start; i < m; i += step) {
        for (size_t j = 0; j < n; j++) {
            res_matr[i][j] += matr_1[i][q2] * matr_2[q2][j];
        }
    }
}
```

3.4 Тестовые данные

Программа должна корректно умножать матрицы при следующих входных данных при количестве рабочих потоков от 1 до 128 (1, 2, 4, 8, 16, ... , 128):

1) матрицы 1×1:

$$[2] \times [3] = [6];$$

2) умножение на нулевую матрицу:

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \times \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix};$$

3) умножение на единичную матрицу:

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \times \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix};$$

4) умножение(матрицы 2×2) на матрицу с положительными числами:

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \times \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} = \begin{bmatrix} 7 & 10 \\ 15 & 22 \end{bmatrix};$$

5) умножение (матрицы 2×2) на матрицу отрицательными числами:

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \times \begin{bmatrix} -1 & -2 \\ -3 & -4 \end{bmatrix} = \begin{bmatrix} -7 & -10 \\ -15 & -22 \end{bmatrix};$$

6) умножение (матрицы 3×3) на матрицу с положительными числами:

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \times \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} = \begin{bmatrix} 30 & 36 & 42 \\ 66 & 81 & 96 \\ 102 & 126 & 150 \end{bmatrix};$$

7) умножение (матрицы 3×3) на матрицу с отрицательными числами:

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \times \begin{bmatrix} -1 & -2 & -3 \\ -4 & -5 & -6 \\ -7 & -8 & -9 \end{bmatrix} = \begin{bmatrix} -30 & -36 & -42 \\ -66 & -81 & -96 \\ -102 & -126 & -150 \end{bmatrix};$$

8) умножение на прямоугольную матрицу с нечётным количеством столбцов:

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \times \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} = \begin{bmatrix} 22 & 28 \\ 49 & 64 \end{bmatrix};$$

9) умножение на прямоугольную матрицу с чётным количеством столбцов:

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \end{bmatrix} \times \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \\ 7 & 8 \end{bmatrix} = \begin{bmatrix} 50 & 60 \\ 114 & 140 \end{bmatrix};$$

10) умножение матриц с большими числами:

$$\begin{bmatrix} 1000 & 2000 & 3000 \\ 4000 & 5000 & 6000 \\ 7000 & 8000 & 9000 \end{bmatrix} \times \begin{bmatrix} 1000 & 2000 & 3000 \\ 4000 & 5000 & 6000 \\ 7000 & 8000 & 9000 \end{bmatrix} = \begin{bmatrix} 30000000 & 36000000 & 42000000 \\ 66000000 & 81000000 & 96000000 \\ 102000000 & 126000000 & 150000000 \end{bmatrix};$$

Все тесты успешно пройдены.

Вывод

В данном разделе были рассмотрены требования к программному обеспечению, в качестве средств реализации выбраны язык C++ и фреймворк QT версии 5.13.1, приведён листинг программы и тестовые данные.

4 Исследовательский раздел

4.1 Примеры работы

На рис. 9, 10, 11 и 12 приведены примеры работы программы для различных входных данных.

```
Enter number of matrix_1 rows: -2
Incorrect input. Try again: wfdsfs
Incorrect input. Try again: 2
Enter number of matrix_1 columns: 2
matr[0][0] = poore
Incorrect input. Try again: 1
matr[0][1] = 2
matr[1][0] = 3
matr[1][1] = 4
standard:
1 2
3 4
Enter number of matrix 2 columns: -5
Incorrect input. Try again: 2
matr[0][0] = 1
matr[0][1] = 0
matr[1][0] = 0
matr[1][1] = 1
Enter count threads: -2
Incorrect input. Try again: 4
Matrix 1:
1 2
3 4
Matrix 2:
1 0
0 1
Result of multiplication by Vinograd is optimized:
1 2
3 4
```

Рис. 9: Пример работы программы для некорректного ввода

```
Enter number of matrix_1 rows: 2
Enter number of matrix_1 columns: 2
matr[0][0] = 1
matr[0][1] = 2
matr[1][0] = 3
matr[1][1] = 4
standard:
1 2
3 4
Enter number of matrix 2 columns: 2
matr[0][0] = 1
matr[0][1] = 2
matr[1][0] = 3
matr[1][1] = 4
Enter count threads: 3
Matrix 1:
1 2
3 4
Matrix 2:
1 2
3 4
Result of multiplication by Vinograd is optimized:
7 10
15 22
```

Рис. 10: Пример работы программы для матриц четной размерности (2×2)

```
matr[0][0] = 1
matr[0][1] = 2
matr[1][0] = 3
matr[1][1] = 4
matr[2][0] = 5
matr[2][1] = 6
standard:
1 2
3 4
5 6
Enter number of matrix 2 columns: 3
matr[0][0] = 1
matr[0][1] = 2
matr[0][2] = 3
matr[1][0] = 4
matr[1][1] = 5
matr[1][2] = 6
Enter count threads: 1
Matrix 1:
1 2
3 4
5 6
Matrix 2:
1 2 3
4 5 6
Result of multiplication by Vinograd is optimized:
9 12 15
19 26 33
29 40 51
```

Рис. 11: Пример работы программы для матриц нечетной размерности (2×3 и 3×2

```
matr[2][1] = 8
matr[2][2] = 9
standard:
1 2 3
4 5 6
7 8 9
Enter number of matrix 2 columns: 3
matr[0][0] = 1
matr[0][1] = 2
matr[0][2] = 3
matr[1][0] = 4
matr[1][1] = 5
matr[1][2] = 6
matr[2][0] = 7
matr[2][1] = 8
matr[2][2] = 9
Enter count threads: 1024
Matrix 1:
1 2 3
4 5 6
7 8 9
Matrix 2:
1 2 3
4 5 6
7 8 9
Result of multiplication by Vinograd is optimized:
30 36 42
66 81 96
102 126 150
```

Рис. 12: Пример работы программы для большого количества рабочих потоков

4.2 Постановка эксперимента

Необходимо выполнить следующие замеры времени:

1) Сравнить время умножения матриц оптимизированным алгоритмом Винограда для последовательной реализации алгоритма и для параллельной реализации с одним рабочим потоком:

   1.1) сравнение на матрицах размером от 100×100 до 1000×1000 с шагом 100

   1.2) сравнение на матрицах размером от 101×101 до 1001×1001 с шагом 100

2) Сравнить время умножения матриц оптимизированным алгоритмом Винограда для параллельной реализации с количеством рабочих потоков от 1 до 32 (количество потоков меняется как геометрическая прогрессия с шагом 2).

Каждый замер производится 10 раз, а затем находится среднее значение.

4.3 Сравнительный анализ на материале экспериментальных данных

Замеры были произведены на 4-ядерном процессоре Intel Core i7 с тактовой частотой 2,2 ГГц с 8-ю логическими потоками, оперативная память — 16 ГБ.

В таблицах 1 и 2 приведены результаты замеров времени выполнения для последовательной реализации и параллельной реализации с одним рабочим потоком.

Из этого следует, что последовательная и параллельная (с 1 рабочим потоком) реализации приблизительно одинаковы по скорости выполнения, но последовательная реализация немного быстрее (на 0.0015% быстрее для матриц размером 1000×1000 и на 0.0021% быстрее для матриц размером 1001×1001). Это связано с тем, что последовательной реализации не нужно тратить время на выделение рабочего потока.

Таблица 1: Время выполнение алгоритма для последовательной и параллельной (1 рабочий поток) реализаций на матрицах четной размерности

Размерность матрицы	Последовательная реализация (мс)	Параллельная (1 рабочий поток) реализация (мс)
100×100	7.3	8.2
200×200	64.1	64.5
300×300	222.9	222.4
400×400	555	556.2
500×500	1121.1	1126.3
600×600	2002.9	1989.9
700×700	3285.1	3294.6
800×800	5237.7	5250.4
900×900	7832.5	7839.5
1000×1000	12755.1	12774.8

Таблица 2: Время выполнение алгоритма для последовательной и параллельной (1 рабочий поток) реализаций на матрицах нечетной размерности

Размерность матрицы	Последовательная реализация (мс)	Параллельная (1 рабочих поток) реализация (мс)
101×101	7.5	8.8
201×201	65.3	65.8
301×301	224.4	225.8
401×401	559.8	561.3
501×501	1128.8	1128.6
601×601	1996.8	2000.6
701×701	3324.2	3327.3
801×801	5235.2	5260.7
901×901	7869.1	7885.3
1001×1001	12691	12719.2

На рис. 13 и 14 приводятся графики времени выполнения оптимизированного алгоритма Винограда для параллельной реализации с количеством рабочих потоков от 1 до 32 на матрицах четной и неченой размерности.

Как видно из этих зависимостей, ощутимой разницы, между временем выполнения алгоритма при четной и нечетной размерностях матрицы нет. При двух рабочих потоках произведение матриц выполняется быстрее по сравнению с выполнением на одном рабочем потоке в 2,09 раз, на четырех рабочих потоках - быстрее в 3,17 раз, на восьми рабочих потоках - в 3,82 раз (все соотношения вычислены для матриц размера 1000×1000). На 16 и 32 рабочих потоках время выполнения немного ухудшается, по сравнению с временем выполнения на 8 рабочих потоках. Максимальная производительность достигается на 8 рабочих потоках, т. к. на тестируемом компьютере имеется 8 логических потоков, т. е. при 8 рабочих потоках они все загружаются, а при большем количестве рабочих потоков, на каждый логический поток приходится уже несколько рабочих и производительность падает (тратится время на создание новых рабочих потоков, но вычисления будут производиться с той же скоростью, что и при 8 рабочих потоках).

Вывод

Эксперименты замера времени показали, что при последовательной и параллельной (с одним рабочим потоком) реализациях оптимизированного алгоритма Винограда совсем немного выигрывает последовательная реализация (в ней не тратится время на выделение рабочего потока). На матрицах размером 1000×1000 последовательная реализация на 0.0015% быстрее параллельной.

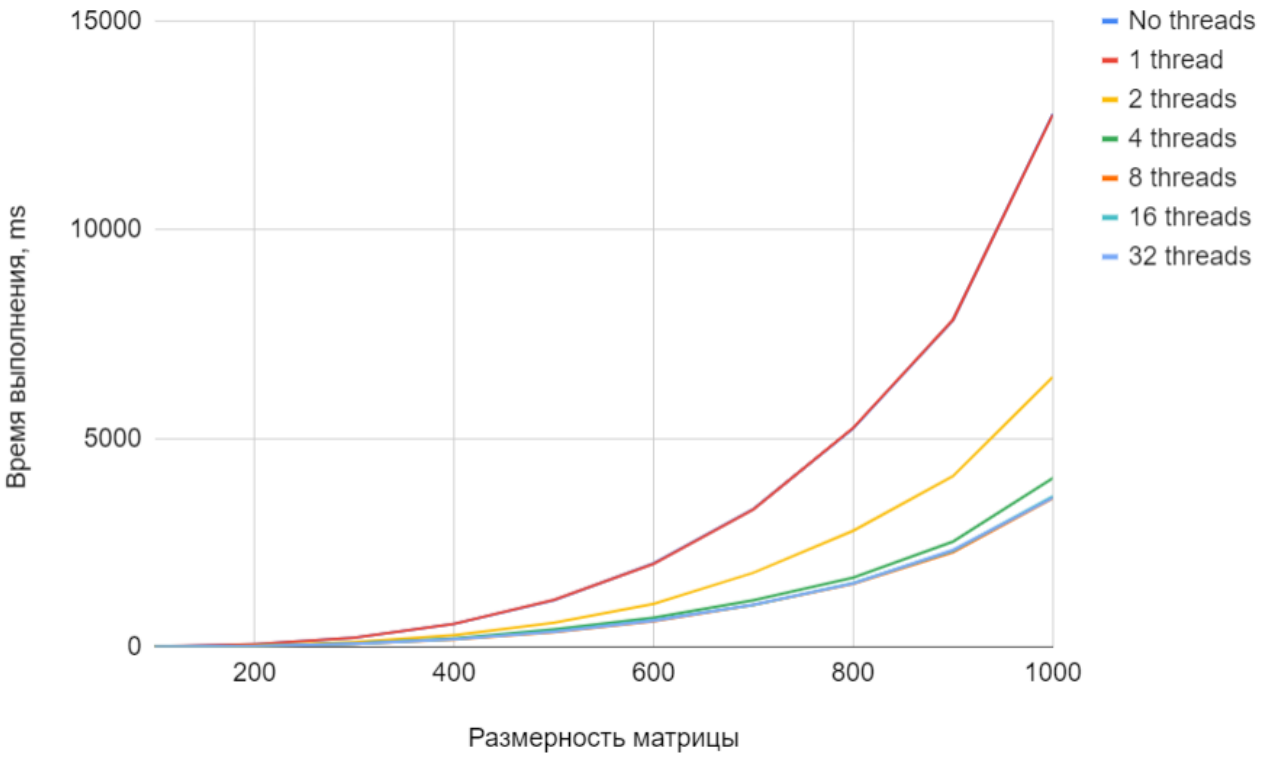

Рис. 13: Сравнение времени выполнения алгоритма для параллельной реализации на матрицах четной размерности

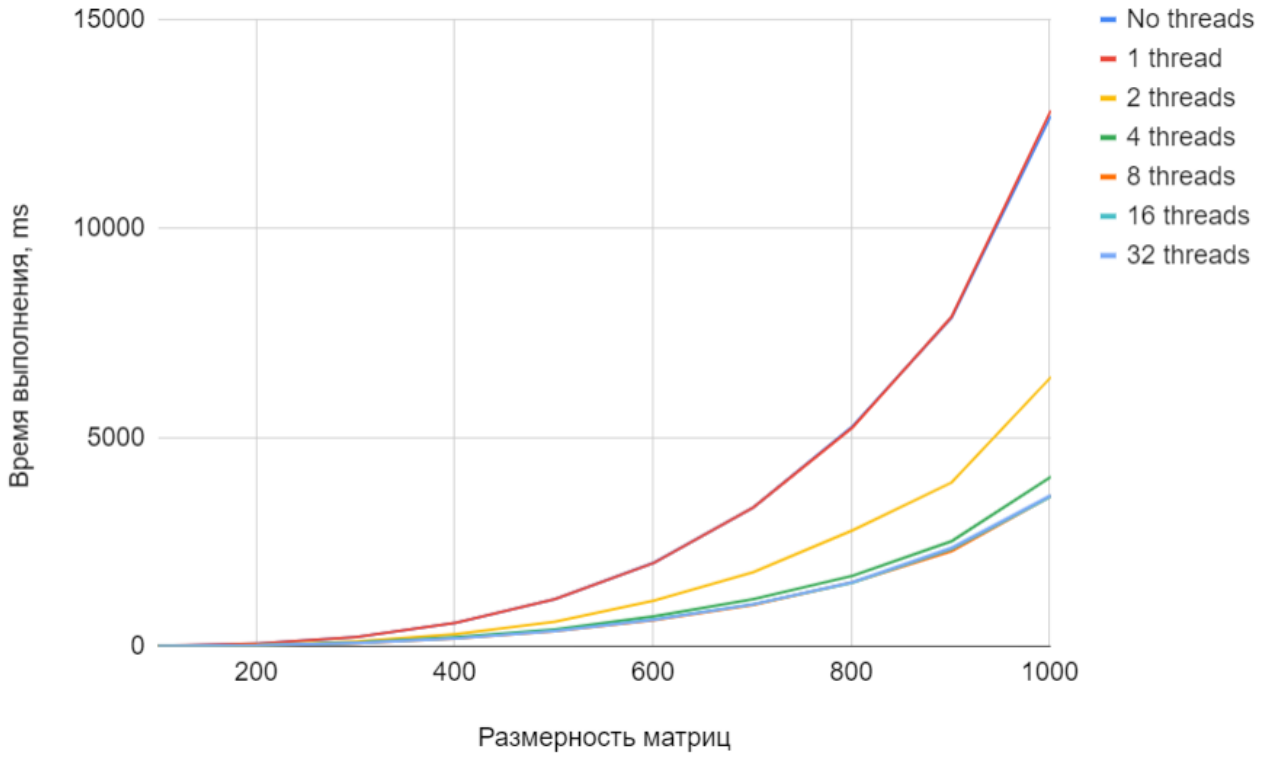

Рис. 14: Сравнение времени выполнения алгоритма для параллельной реализации на матрицах нечетной размерности

При сравнении замеров времени для параллельной реализации алгоритма с 1-м, 2-мя, 4-мя, 8-ю, 16-ю и 32-мя рабочими потоками выяснилось, что максимальная производительность достигается на 8-ми рабочих потоках , что равно количеству логических потоков компьютера,

на котором производились замеры. Выполнение алгоритма на 8-ми рабочих потоках быстрее в 3,82 раз, по сравнению с выполнением на 1-м потоке для матриц размера 1000×1000. При большем количестве рабочих потоков происходит небольшое падение производительности (тратится время на создание новых рабочих потоков, но вычисления будут производиться с той же скоростью, что и при 8 рабочих потоках).

Заключение

В ходе работы было выполнено следующее:

1) изучен алгоритм умножения матриц по Винограду;
2) оптимизирован алгоритма Винограда
3) реализована многопоточность для оптимизированного алгоритма Винограда
3) применен метод динамического программирования для реализации указанных алгоритмов;
4) приобретены практические навыки реализации указанных алгоритмов: оптимизированного алгоритма Винограда, разбитого на потоки;
5) проведен сравнительный анализ по затрачиваемым ресурсам (времени) при разном количестве рабочих потоков;
7) описаны и обоснованы полученные результаты в отчете о выполненной лабораторной работе, выполненного как расчётно-пояснительная записка к работе.

Эксперименты замера времени показали, что при последовательной и параллельной (с одним рабочим потоком) реализациях оптимизированного алгоритма Винограда совсем немного выигрывает последовательная реализация (в ней не тратится время на выделение рабочего потока). На матрицах размером 1000×1000 последовательная реализация на 0.0015% быстрее параллельной.

При сравнении замеров времени для параллельной реализации алгоритма с 1-м, 2-мя, 4-мя, 8-ю, 16-ю и 32-мя рабочими потоками выяснилось, что максимальная производительность достигается на 8-ми рабочих потоках , что равно количеству логических потоков компьютера, на котором производились замеры. Выполнение алгоритма на 8-ми рабочих потоках быстрее в 3,82 раз, по сравнению с выполнением на 1-м потоке для матриц размера 1000×1000. При большем количестве рабочих потоков происходит небольшое падение производительности (тратится время на создание новых рабочих потоков, но вычисления будут производиться с той же скоростью, что и при 8 рабочих потоках).

Список литературы

[1] Дж. Макконнелл. Анализ алгоритмов. Активный обучающий подход.- М.:Техносфера, 2009.

[2] Матрицы и определители [Электронный ресурс]. – Режим доступа: http://matematika.electrichelp.ru/matricy-i-opredeliteli/, свободный – (07.10.2019)