

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

# Содержание

<b>Введение</b>	<b>3</b>
<b>1 Аналитический раздел</b>	<b>4</b>
1.1 Описание алгоритмов . . . . .	4
1.1.1 Стандартный алгоритм . . . . .	4
1.1.2 Алгоритм Винограда . . . . .	4
Вывод . . . . .	5
<b>2 Конструкторский раздел</b>	<b>6</b>
2.1 Разработка алгоритмов . . . . .	6
2.1.1 Неоптимизированный алгоритм Винограда . . . . .	6
2.1.2 Оптимизация алгоритма Винограда . . . . .	6
2.1.3 Реализация многопоточности . . . . .	7
Вывод . . . . .	12
<b>3 Технологический раздел</b>	<b>13</b>
3.1 Требования к программному обеспечению . . . . .	13
3.2 Средства реализации . . . . .	13
3.3 Листинг программы . . . . .	13
3.4 Тестовые данные . . . . .	15
Вывод . . . . .	16
<b>4 Исследовательский раздел</b>	<b>17</b>
4.1 Примеры работы . . . . .	17
4.2 Постановка эксперимента . . . . .	19
4.3 Сравнительный анализ на материале экспериментальных данных . . . . .	19
Вывод . . . . .	21
<b>Заключение</b>	<b>22</b>
<b>Список литературы</b>	<b>23</b>

## 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:

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

где  $a_{ij} (i = 1, \dots, m; j = 1, \dots, n)$  — это элементы матрицы  $A$ . Первый индекс  $i$  — это номер строки, второй индекс  $j$  — это номер столбца, на пересечении которых расположен элемент  $a_{ij}$  [3].

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]:

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

### 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:

$$V \cdot W = v_1 w_1 + v_2 w_2 + v_3 w_3 + v_4 w_4.$$

Это равенство можно представить так:

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

Такой подход позволяет вычислять  $-v_1 v_2 - v_3 v_4$  и  $-w_1 w_2 - w_3 w_4$  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[?].

## **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 Неоптимизированный алгоритм Винограда

На рис. 1 приведена схема неоптимизированного алгоритма Винограда.

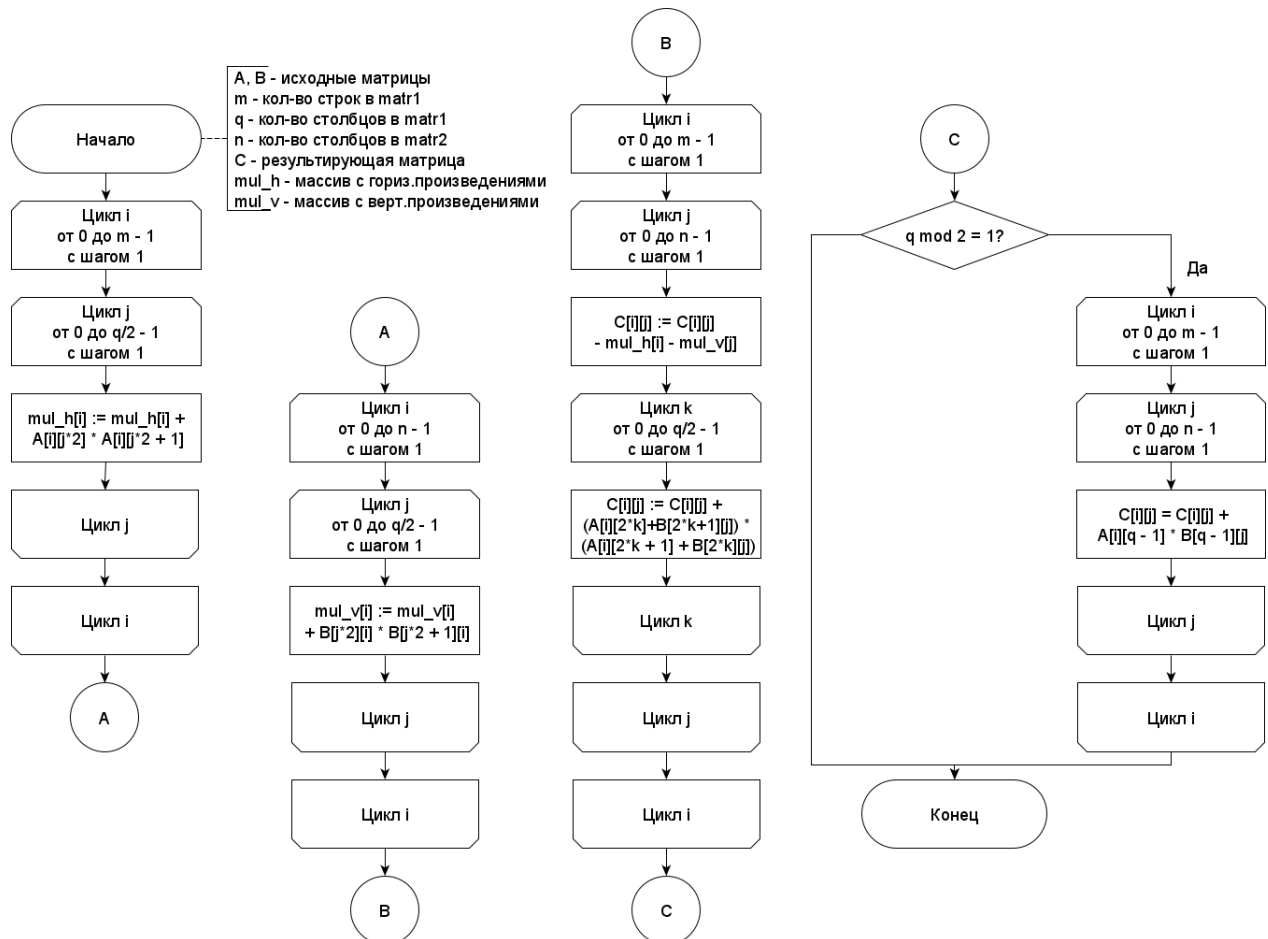


Рис. 1: Схема алгоритма умножения матриц по Винограду (часть 1)

#### 2.1.2 Оптимизация алгоритма Винограда

Заполнение массива под горизонтальные (вертикальные) произведения:

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

Тройной цикл:

- 1) замена  $=$  на  $+=$ ;
- 2) замена в цикле по  $q$  шага цикла с 1 на 2  $\Rightarrow$  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. 2, 3, 4, 5 и 6 представлены схемы оптимизированного алгоритма Винограда, разбитого на потоки.

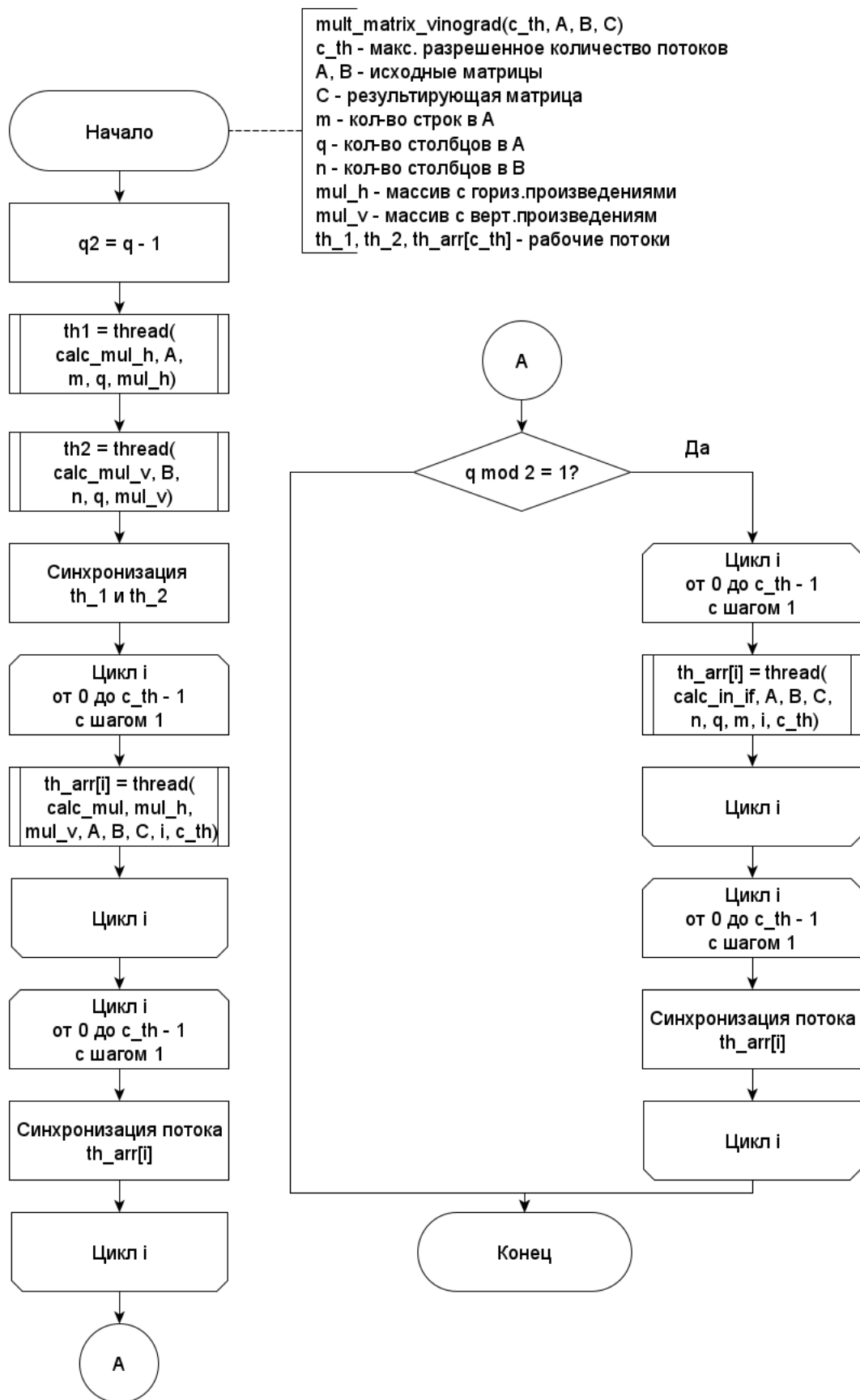


Рис. 2: Схема основной функции



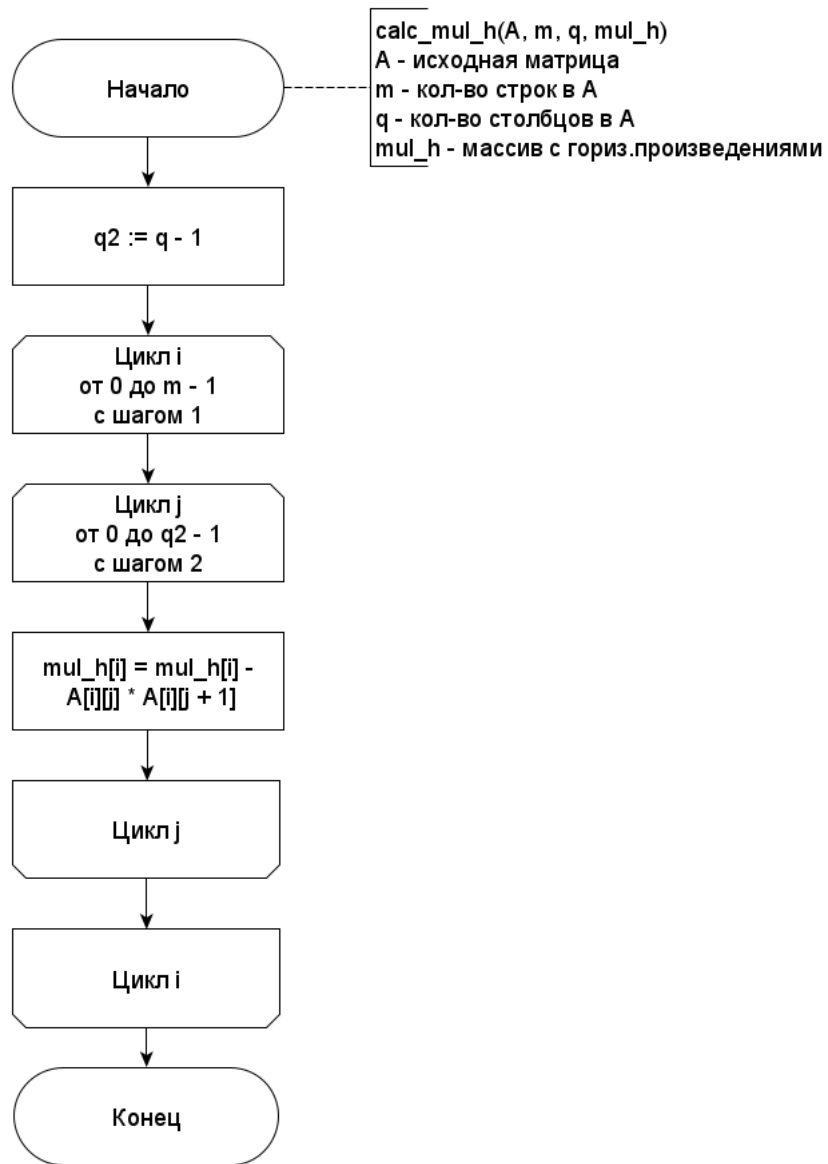


Рис. 3: Схема функции, вычисляющей горизонтальные произведения

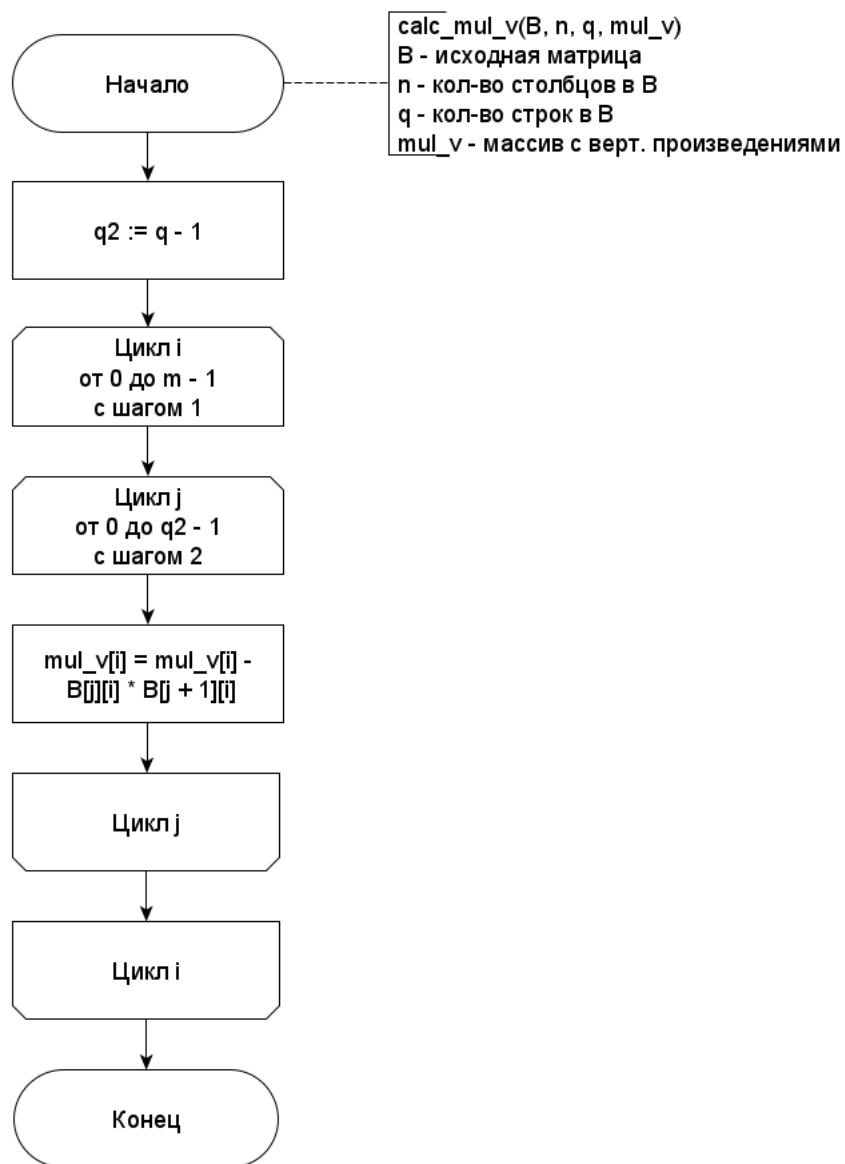


Рис. 4: Схема функции, вычисляющей вертикальные произведения

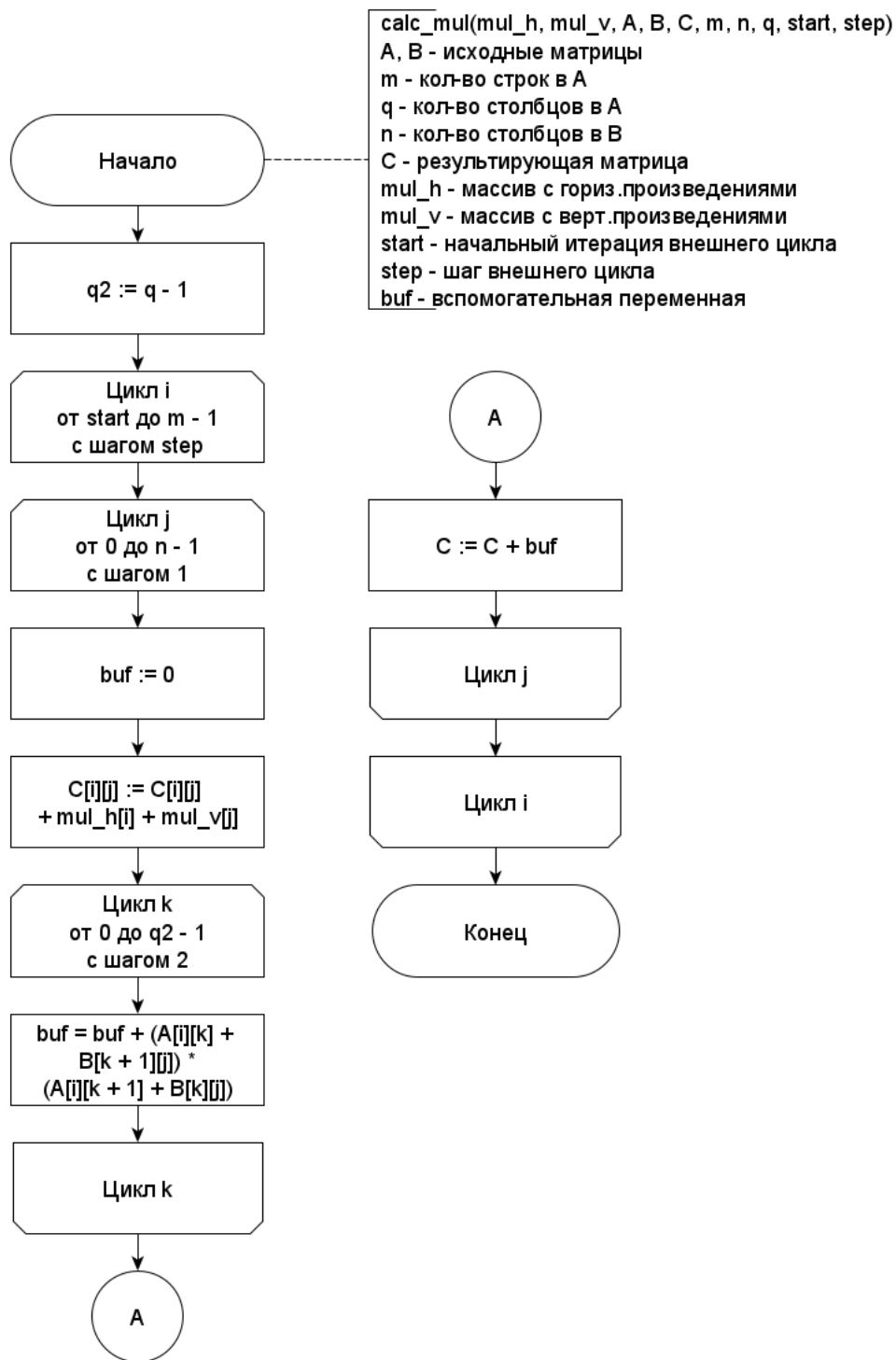


Рис. 5: Схема функции вычислений в тройном цикле

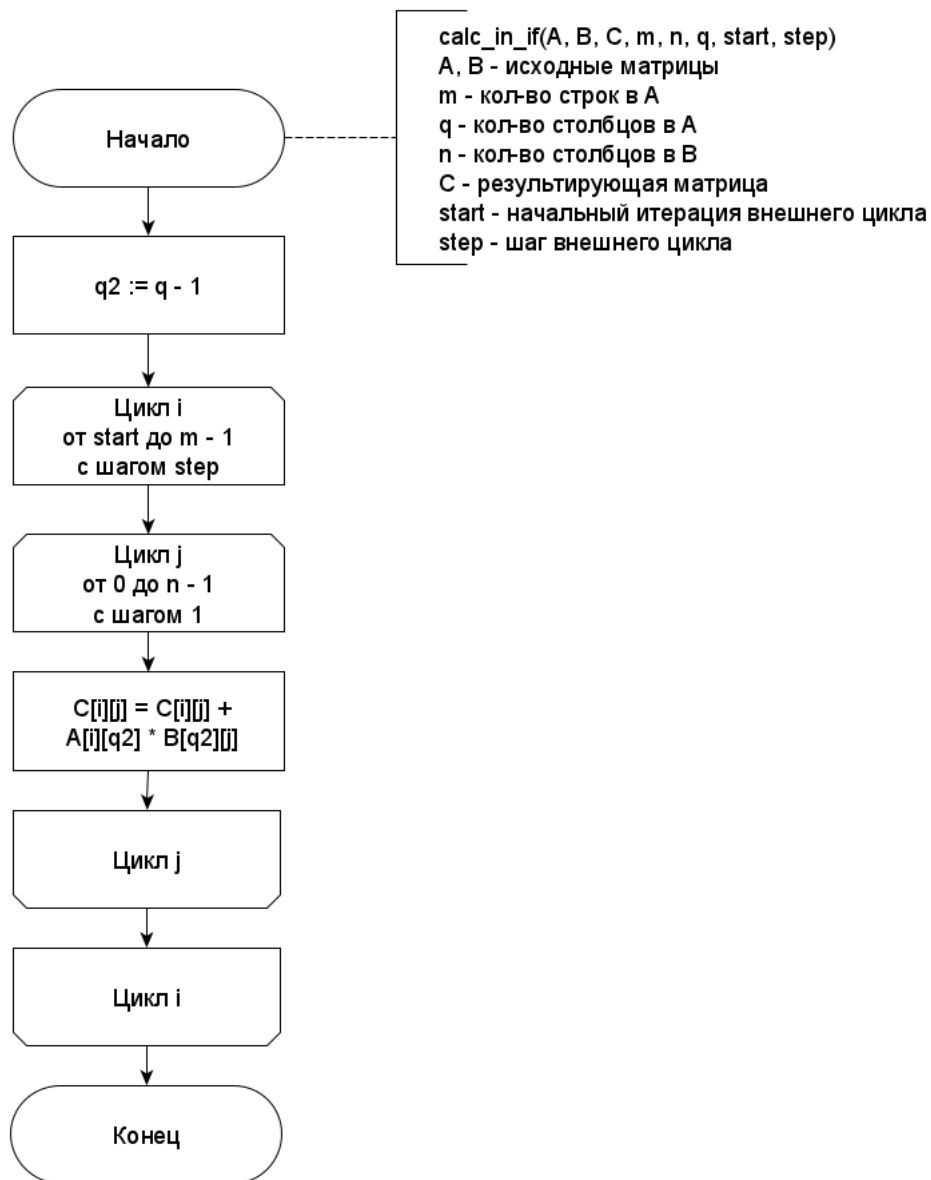


Рис. 6: Схема функции вычислений внутри условного перехода

## 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  $M \times Q$  и  $Q \times N$ , заполненные целыми числами
- 3) целое число - максимальное количество рабочих потоков

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

На рис. 7 приведена функциональная схема вычисления произведения матриц по оптимизированному алгоритму Винограда с разбиением на потоки.

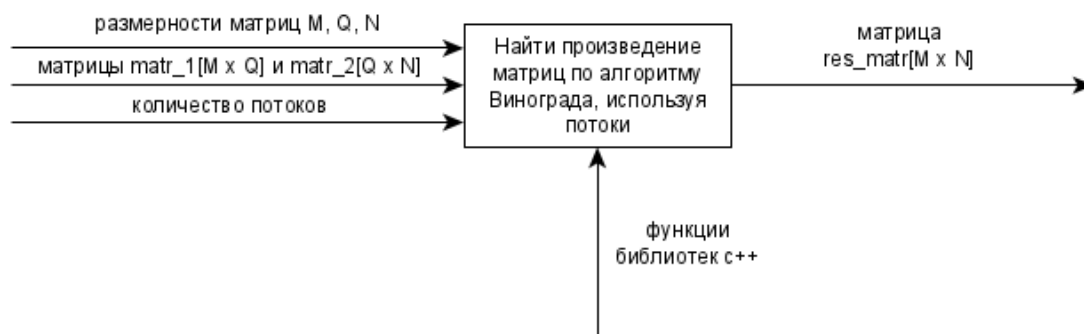


Рис. 7: Функциональная схема вычисления произведения матриц по оптимизированному алгоритму Винограда с разбиением на потоки

#### 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 использовалась среда разработки `CLion`.

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 оптимизированному алгоритму Винограду с разделением на потоки. В листингах 2, 3, 4 и 5 представлены функции, которые обрабатываются на рабочих потоках.

Листинг 1: Основная функция

```
1 | void mult_matrix_vinograd_optimiz(int count_th, Matrix matr_1, Matrix matr_2,
```

```

2                                     Matrix &res_matr) {
3     size_t m = matr_1.size();
4     size_t q = matr_1[0].size();
5     size_t q2 = q - 1;
6     size_t n = matr_2[0].size();
7
8     Vector mul_h(m, 0);
9     Vector mul_v(n, 0);
10
11     std::thread thread_1(calc_mul_h, std::ref(matr_1), m, q2,
12                           std::ref(mul_h));
13     std::thread thread_2(calc_mul_v, std::ref(matr_2), n, q2,
14                           std::ref(mul_v));
15     thread_1.join();
16     thread_2.join();
17
18     std::vector<std::thread> thread_arr(count_th);
19
20     for(int i = 0; i < count_th; i++) {
21         thread_arr[i] = std::thread(calc_mult, std::ref(mul_h),
22                                     std::ref(mul_v), std::ref(matr_1),
23                                     std::ref(matr_2),
24                                     std::ref(res_matr), i, count_th);
25     }
26     for(int i = 0; i < count_th; i++) {
27         thread_arr[i].join();
28     }
29
30     if (q % 2 == 1) {
31         for(int i = 0; i < count_th; i++) {
32             thread_arr[i] = std::thread(calc_in_if, std::ref(matr_1), std::ref(matr_2),
33                                         std::ref(res_matr), i, count_th);
34         }
35         for(int i = 0; i < count_th; i++) {
36             thread_arr[i].join();
37         }
38     }
39 }

```

Листинг 2: Функция вычисления горизонтальных произведений

```

1 using Matrix = std::vector<std::vector<int>>>;
2 using Vector = std::vector<int>;
3
4 void calc_mul_h(Matrix &matr_1, size_t m, size_t q2, Vector &mul_h) {
5     for(size_t i = 0; i < m; i++) {
6         for (size_t j = 0; j < q2; j+=2) {
7             mul_h[i] -= matr_1[i][j] * matr_1[i][j + 1];
8         }
9     }
10 }

```

Листинг 3: Функция вычисления вертикальных произведений

```

1 void calc_mul_v(Matrix &matr_2, size_t n, size_t q2, Vector &mul_v) {
2     for(size_t i = 0; i < n; i++) {
3         for (size_t j = 0; j < q2; j+=2) {
4             mul_v[i] -= matr_2[j][i] * matr_2[j + 1][i];
5         }
6     }
7 }

```

Листинг 4: Функция вычислений в тройном цикле

```

1 void calc_mult(Vector &mul_h, Vector &mul_v, Matrix &matr_1, Matrix &matr_2,
2               Matrix &res_matr, size_t start, size_t step) {

```

```

3   size_t m = matr_1.size();
4   size_t q2 = matr_1[0].size() - 1;
5   size_t n = matr_2[0].size();
6   for(size_t i = start; i < m; i += step) {
7       for (size_t j = 0; j < n; j++) {
8           res_matr[i][j] = mul_h[i] + mul_v[j];
9           int buf = 0;
10          for(size_t k = 0; k < q2; k+=2) {
11              buf += (matr_1[i][k] + matr_2[k + 1][j]) *
12                  (matr_1[i][k + 1] + matr_2[k][j]);
13          }
14          res_matr[i][j] += buf;
15      }
16  }
17 }

```

Листинг 5: Функция дополнительных вычислений для матриц нечетной размерности

```

1   void calc_in_if(Matrix &matr_1, Matrix &matr_2, Matrix &res_matr,
2       size_t start, size_t step) {
3       size_t m = matr_1.size();
4       size_t q2 = matr_1[0].size() - 1;
5       size_t n = matr_2[0].size();
6       for(size_t i = start; i < m; i += step) {
7           for (size_t j = 0; j < n; j++) {
8               res_matr[i][j] += matr_1[i][q2] * matr_2[q2][j];
9           }
10      }
11 }

```

### 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  $1 \times 1$ :

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

- 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  $2 \times 2$ ) 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  $2 \times 2$ ) 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  $3 \times 3$ ) 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  $3 \times 3$ ) 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 среда разработки CLion, приведён листинг программы и тестовые данные.



## 4 Исследовательский раздел

### 4.1 Примеры работы

На рис. 8, 9, 10 и 11 приведены примеры работы программы для различных входных данных.

```
Enter number of matrix_1 rows:-1
Incorrect input. Try again:asdf
Incorrect input. Try again:2
Enter number of matrix_1 columns:
2
  matr[0][0] =asdf
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:-12
Incorrect input. Try again:2
  matr[0][0] =1
  matr[0][1] =0
  matr[1][0] =0
  matr[1][1] =1
Enter
count threads:-3
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. 8: 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
sta
ndard:
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. 9: Пример работы программы для матриц четной размерности ( $2 \times 2$ )

```

matr[0][0] =1
matr[0][1] =2
matr[1][0] =3
matr[1][1] =4
mat
r[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 co
unt 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. 10: Пример работы программы для матриц нечетной размерности ( $2 \times 3$  и  $3 \times 2$ )

```

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. 11: 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  $100 \times 100$  до  $1000 \times 1000$  с шагом 100
  - 1.2) сравнение на матрицах размером от  $101 \times 101$  до  $1001 \times 1001$  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 3,5 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 параллельная реализация немно-

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)
$100 \times 100$	18.3	9.6
$200 \times 200$	101.0	79.3
$300 \times 300$	298.7	262.6
$400 \times 400$	723.0	649.0
$500 \times 500$	1426.0	1265.0
$600 \times 600$	2424.3	2235.6
$700 \times 700$	3946.0	3711.0
$800 \times 800$	6145.7	5574.0
$900 \times 900$	9212.7	8430.6
$1000 \times 1000$	14763.7	13261.6

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)
$101 \times 101$	10.0	10.3
$201 \times 201$	84.0	80.0
$301 \times 301$	302.0	299.6
$401 \times 401$	699.3	653.0
$501 \times 501$	1387.0	1299.3
$601 \times 601$	2412.3	2282.6
$701 \times 701$	3948.0	3779.0
$801 \times 801$	6071.6	5711.0
$901 \times 901$	9240.0	8312.3
$1001 \times 1001$	14919.6	13326.6

го быстрее.

На рис. 12 и 13 приводятся графики времени выполнения оптимизированного алгоритма Винограда для параллельной реализации с количеством рабочих потоков от 1 до 32 на матрицах четной и нечетной размерности.

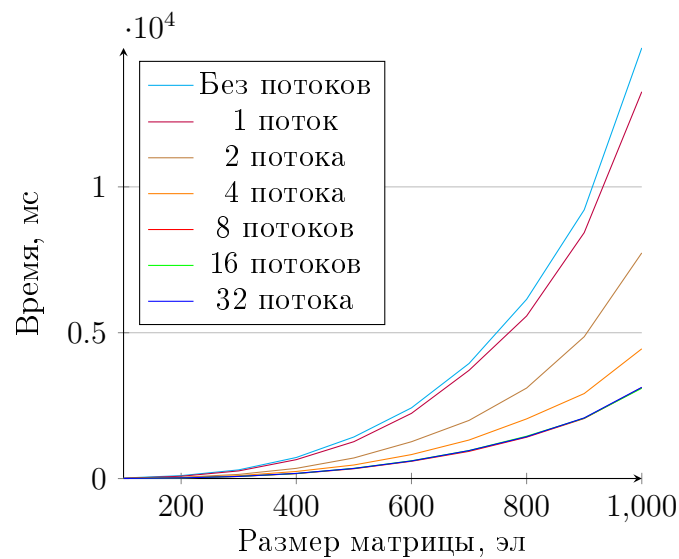


Рис. 12: Сравнение времени выполнения алгоритма для параллельной реализации на матрицах четной размерности

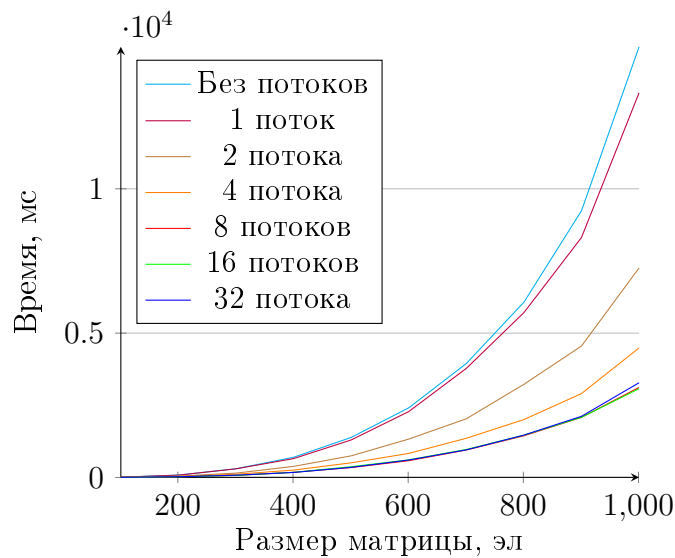


Рис. 13: Сравнение времени выполнения алгоритма для параллельной реализации на матрицах нечетной размерности

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  $1000 \times 1000$ ). На 16 и 32 рабочих потоках время выполнения совпадает 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 может снизиться (тратится время на создание новых рабочих потоков, но вычисления будут производиться с той же скоростью, что и при 8 рабочих потоках).

## Вывод

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  $1000 \times 1000$ .

## Заключение

В ходе работы было выполнено следующее:

- 1) изучен алгоритм умножения матриц по Винограду;
- 2) оптимизирован алгоритм 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  $1000 \times 1000$  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  $1000 \times 1000$ .

## Список литературы

- [1] Макконнелл Дж. Основы современных алгоритмов = Analysis of Algorithms: An Active Learning Approach / Под ред. С. К. Ландо. — М.: Техносфера, 2004. — С. 72-76. — ISBN 5-94836-005-9.
- [2] Матрицы и определители [Электронный ресурс]. — Режим доступа: <http://matematika.electrichelp.ru/matrixy-i-opredeliteli/>, свободный — (16.02.2020)
- [3] Язык программирования с++ [Электронный ресурс]. — Режим доступа: <https://isocpp.org/>, свободный — (16.02.2020)
- [4] Кнут Д. Э. 5.2.2 Обменная сортировка // Искусство программирования. Том 3. Сортировка и поиск = The Art of Computer Programming. Volume 3. Sorting and Searching / под ред. В. Т. Тертышного (гл. 5) и И. В. Красикова (гл. 6). — 2-е изд. — Москва: Вильямс, 2007. — Т. 3. — 832 с. — ISBN 5-8459-0082-1.
- [5] Левитин А. В. Глава 3. Метод грубой силы: Пузырьковая сортировка // Алгоритмы. Введение в разработку и анализ — М.: Вильямс, 2006. — С. 144–146. — 576 с. — ISBN 978-5-8459-0987-9

