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 № 4

Дисциплина Анализ алгоритмов

Тема Параллельное программирование

Студент Панафидин Егор

Группа ИУ7-51Б

Преподаватель Волкова Л.Л.

Москва — 2020 г.

Оглавление

Введение

Цель работы: изучение методов параллельного программирования на примере реализации многопоточного алгоритма Винограда. Простые оптимизации не всегда помогают увеличить скорость программы, поэтому для решения некоторых задач используют параллелизацию независимых процессов в целях уменьшения времени работы программы. В данной лабораторoной работе требуется применить этот метод и провести сравнительный анализ временных характеристик реализации многопоточного алгоритма Винограда и обычного.

1. Аналитическая часть

Умножение матриц — одна из основных операций над матрицами. Матрица, получаемая в результате операции умножения, называется произведением матриц. Операция умножения двух матриц выполнима только в том случае, если число столбцов в первом сомножителе равно числу строк во втором; в этом случае говорят, что матрицы согласованы. В частности, умножение всегда выполнимо, если оба сомножителя — квадратные матрицы одного и того же порядка. Таким образом, из существования произведения AB вовсе не следует существование произведения BA.

Эти алгоритмы активно применяются во всех областях, применяющих линейную алгебру, такие как:

- компьютерная графика;
- физика;
- экономика;

1.1 Алгоритм Винограда

Рассмотрим два вектора

$$a = (v1, v2, v3, v4), b = (w1, w2, w3, w4) \tag{1.1}$$

Их скалярное произведение равно

$$a \cdot b = a_1 \cdot b_1 + a_2 \cdot b_2 + a_3 \cdot b_3 + a_4 \cdot b_4 \tag{1.2}$$

Равенство (1.2) можно переписать в виде (1.3)

$$a \cdot b = (a_1 + b_2) \cdot (a_2 + b_1) + (a_3 + b_4) \cdot (a_4 + b_3) - a_1 \cdot a_2 - a_3 \cdot a_4 - b_1 \cdot b_2 - b_3 \cdot b_4 \tag{1.3}$$

Может показаться, что выражение (1.4) задает больше работы, чем (1.3): вместо четырех умножений мы насчитываем их шесть, а вместо трех сложений - десять. Менее очевидно, что выражение в правой части последнего равенства допускает предварительную обработку: его части можно вычислить заранее и запомнить для каждой строки первой матрицы и для каждого столбца второй. Это означает, что над предварительно обработанными элементами нам придется выполнять лишь первые два умножения и последующие пять сложений, а также дополнительно два сложения.

2. Конструкторская часть

В данном разделе будут рассмотрены схемы алгоритмов:

- умножения матриц методом Винограда;
- многопоточного умножения матриц методом Винограда;

2.1 Схемы алгоритмов

В данной части будут рассмотрены схемы алгоритмов умножения матриц:

- схема алгоритма стандартного умножения матриц(рис. 2.1);
- схема алгоритма умножения матриц методом Винограда(рис. 2.2);
- схема алгоритма умножения матриц оптимизированным методом Винограда(рис 2.3);

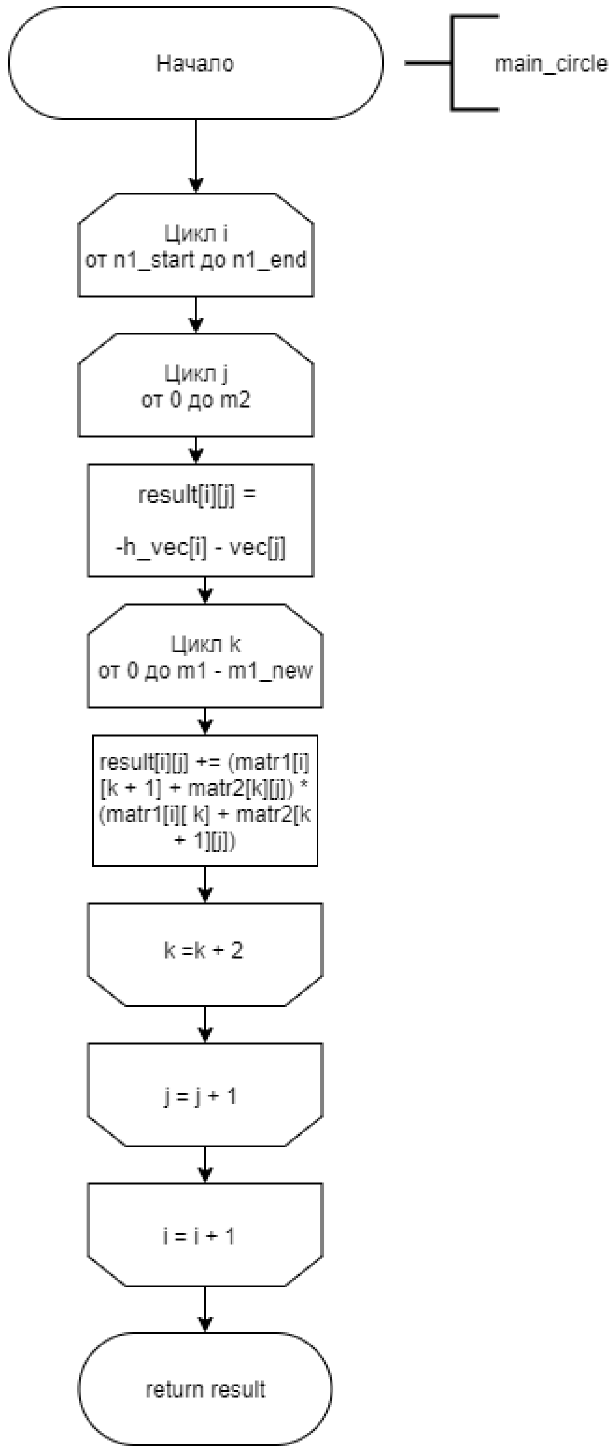

Рис. 2.1: Схема тройного цикла в алгоритме Винограда

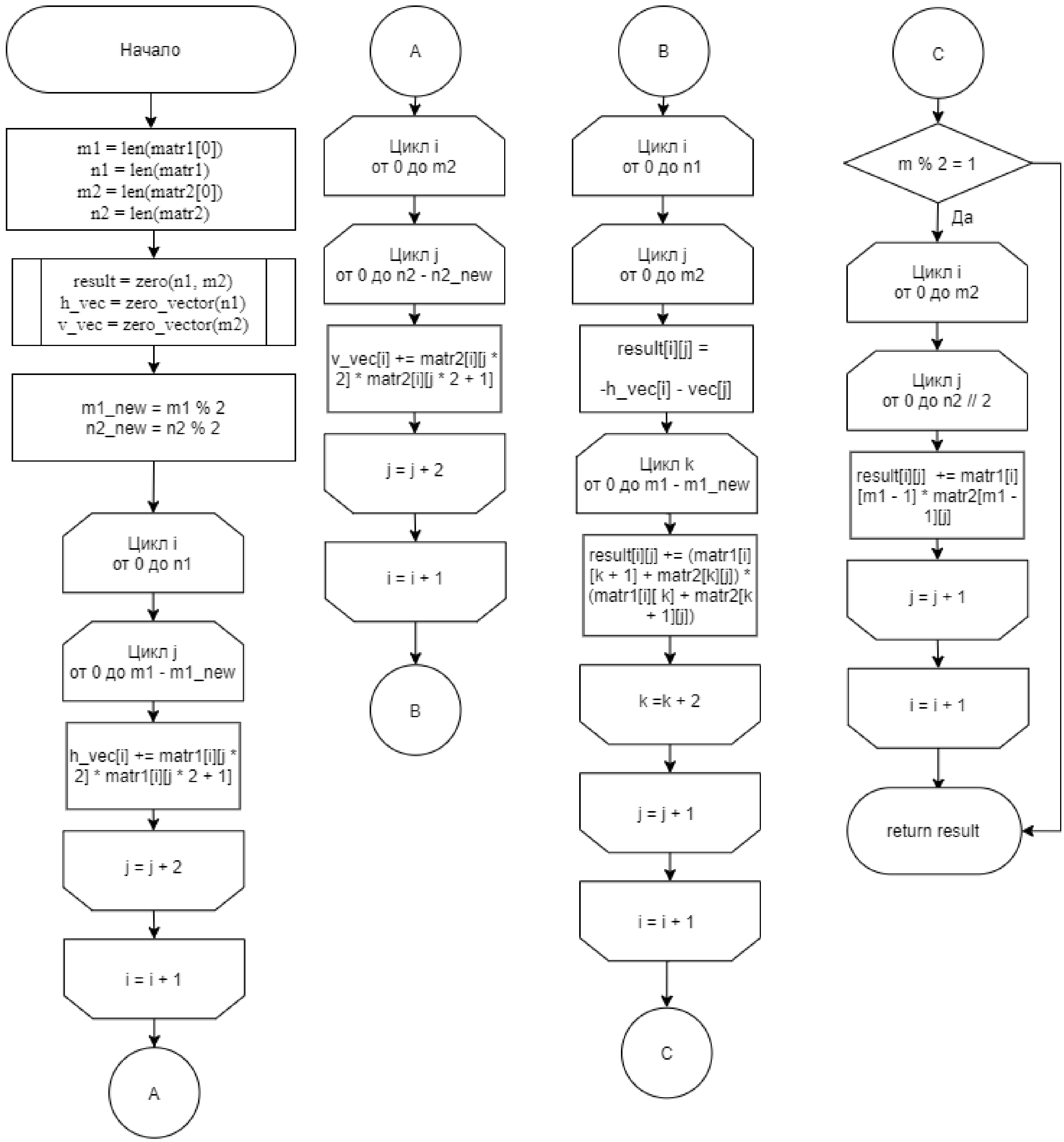

Рис. 2.2: Схема алгоритма умножения матриц методом Винограда

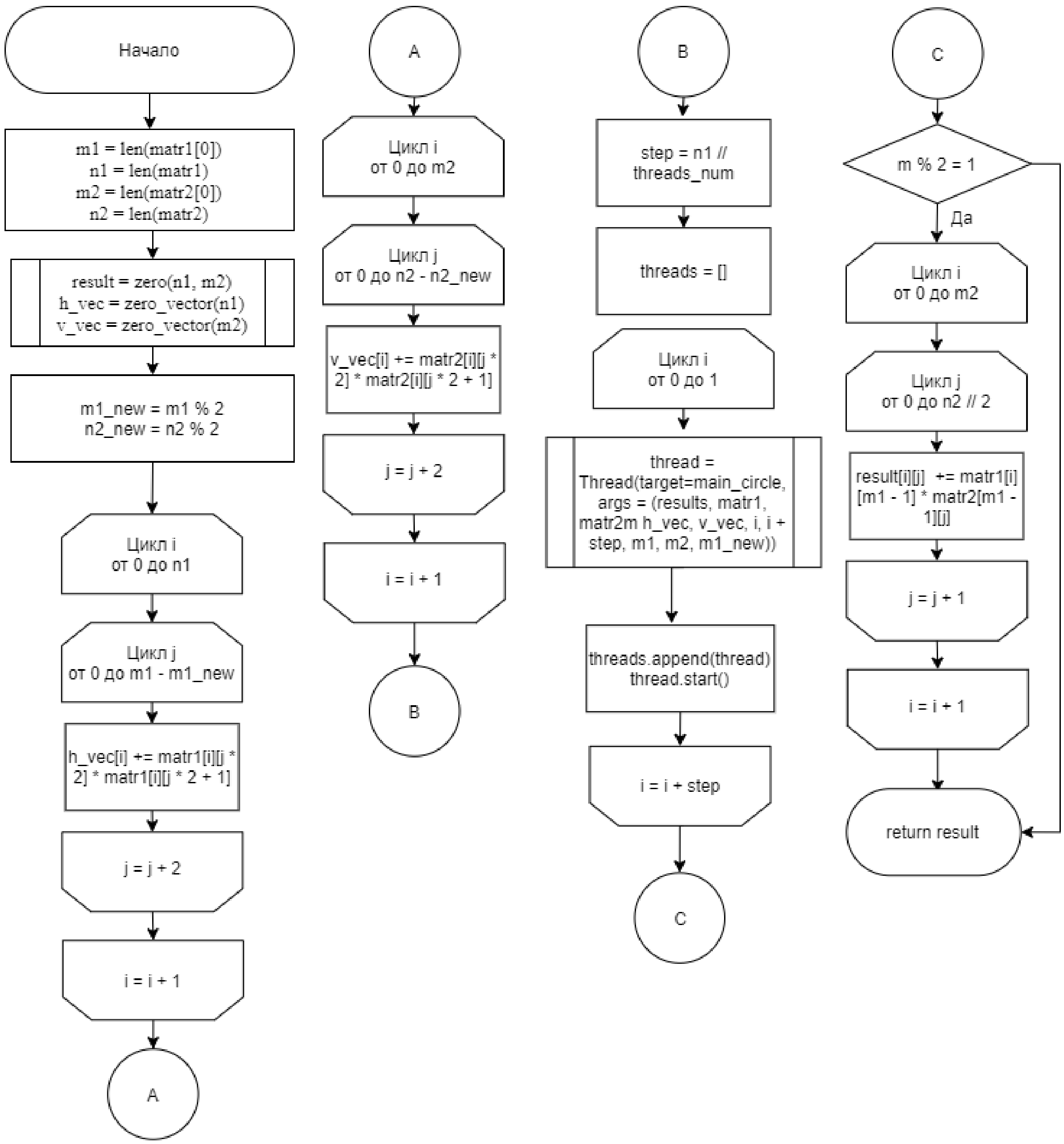

Рис. 2.3: Схема многопоточного алгоритма умножения матриц оптимизированным методом Винограда

3. Технологическая часть

3.1 Выбор ЯП

Для реализации программ был выбран язык Python, потому что он позволяет быстро программировать и имеет достаточно простой для понимания синтаксис и механизм работы.

3.2 Сведения о модулях программы

- func.py - функции для умножения матриц методом Винограда и многопоточным методом Винограда

3.3 Реализация алгоритмов

В данном блоке будут представлены следующие листинги:

- Листинг функции тройного цикла в алгоритме умножения матриц методом Винограда(рис. 3.1);
- Листинг функции умножения матриц методом Винограда(рис. 3.2);
- Листинг функции умножения матриц многопоточным методом Винограда(рис. 3.3);

Листинг 3.1: Функция умножения матриц методом Винограда

```
def vinograd_opt(matr1, matr2):
    start = 0
    m1 = len(matr1[0])
    n1 = len(matr1)
    m2 = len(matr2[0])
    n2 = len(matr2)
    result = zero(n1, m2)
    h_vec = zero_vector(n1)
    v_vec = zero_vector(m2)
    if m1 == n2:
        m1_new = m1 % 2
        n2_new = n2 % 2
        for i in range(n1):
            for j in range(0,m1 - m1_new,2):
                h_vec[i] += matr1[i][j] * matr1[i][j + 1]

        for i in range(m2):
```

```
            for j in range(0,n2 - n2_new,2):
                v_vec[i] += matr2[j][i] * matr2[j + 1][i]

        start = time.clock()
        for i in range(n1):
            for j in range(m2):
                tmp = -h_vec[i] - v_vec[j]
                for k in range(0,m1 - m1_new,2):
                    tmp += (matr1[i][k + 1] + matr2[k][j]) * \
                                       (matr1[i][k] + matr2[k + 1][j])
                result[i][j] = tmp
        if m1_new:
            for i in range(n1):
                for j in range(m2):
                    result[i][j] += matr1[i][m1 - 1] * matr2[m1 - 1][j]

    return result, time.clock() - start
```

Листинг 3.2: Функция для умножения многопоточным методом Винограда

```
def vinograd_opt_parallel(matr1, matr2, threads_num):
    start = 0
    m1 = len(matr1[0])
    n1 = len(matr1)
    m2 = len(matr2[0])
    n2 = len(matr2)
    result = zero(n1, m2)
    h_vec = zero_vector(n1)
    v_vec = zero_vector(m2)
    if m1 == n2:
        m1_new = m1 % 2
        n2_new = n2 % 2
        for i in range(n1):
            for j in range(0,m1 - m1_new,2):
                h_vec[i] += matr1[i][j] * matr1[i][j + 1]

        for i in range(m2):
            for j in range(0,n2 - n2_new,2):
                v_vec[i] += matr2[j][i] * matr2[j + 1][i]

        start = time.clock()
        step = n1 // threads_num
        if step == 0:
            step = n1

        step1 = step
        threads = []

        for i in range(0, n1, step):
            if i + step > n1:
                step1 = n1 - i
            thread = threading.Thread(target=main_cycle, args=(result,matr1,
              matr2, h_vec, v_vec, i, i + step1, m1, m2, m1_new))
```

```
            threads.append(thread)
            thread.start()

        if m1_new:
            for i in range(n1):
                for j in range(m2):
                    result[i][j] += matr1[i][m1 - 1] * matr2[m1 - 1][j]

    return result, time.clock() - start
```

Листинг 3.3: Функция главного цикла умножения матриц методом Винограда

```
def main_cycle(result,matr1, matr2, h_vec, v_vec, n1_start, n1_end, m1, m2,
    m1_new):
    for i in range(n1_start, n1_end, 1):
        for j in range(m2):
            tmp = -h_vec[i] - v_vec[j]
            for k in range(0, m1 - m1_new, 2):
                tmp += (matr1[i][k + 1] + matr2[k][j]) * \
                       (matr1[i][k] + matr2[k + 1][j])
            result[i][j] = tmp
```

3.4 Тесты

В данной секции буду размещены тесты:

- Тест умножения матрицы на вектор(рис. 3.1);
- Тест умножения вектора на вектор(рис. 3.2);
- Тест умножения матриц одинаковых размеров(рис 3.3);
- Тест умножения матриц, состоящих из одного элемента(рис 3.3);
- Тест умножения матриц разных размеров(рис 3.4);

```
Матрица A:
1 2 25 4 55
2 2 3 6 5
6 22 33 44 5
Матрица B:
1 2 3 4 5 6
1 2 3 4 5 6
1 2 3 4 5 6
1 2 3 4 5 6
1 2 3 4 5 6
Результат умножения методом Винограда:
87 174 261 348 435 192
18 36 54 72 90 78
110 220 330 440 550 630
Результат умножения многопоточным методом Винограда:
87 174 261 348 435 192
18 36 54 72 90 78
110 220 330 440 550 630
```

Рис. 3.1: Тест умножения матриц разных размеров

```
Матрица A:
1 2 3 4 5
Матрица B:
10
12
13
14
15
Результат умножения методом Винограда:
204
Результат умножения многопоточным методом Винограда:
204
```

Рис. 3.2: Тест умножения вектора на вектор

```
Матрица A:
1 2 3 4 5
1 2 3 4 5
1 2 3 4 5
1 2 3 4 5
Матрица B:
1
2
3
4
5
Результат умножения методом Винограда:
55
55
55
55
Результат умножения многопоточным методом Винограда:
55
55
55
55
```

Рис. 3.3: Тест умножения матрицы на вектор

```
Матрица A:
1 2 3 4 5
1 2 3 4 5
1 2 3 4 5
1 2 3 4 5
Матрица B:
1 2 3 4 5
2 3 4 5 6
3 4 5 6 7
4 5 6 7 8
5 6 7 8 9
Результат умножения методом Винограда:
55 70 85 100 115
55 70 85 100 115
55 70 85 100 115
55 70 85 100 115
Результат умножения многопоточным методом Винограда:
55 70 85 100 115
55 70 85 100 115
55 70 85 100 115
55 70 85 100 115
```

Рис. 3.4: Тест умножения матриц одинаковых размерностей

4. Исследовательская часть

4.1 Анализ времени работы алгоритмов

Обозначения:

- Vinograd - алгоритм умножения матриц методом Винограда;
- VinogradParallel - алгоритм умножения матриц многопоточным методом Винограда.

Замер времены работы алгоритмов производился 5 раз, а результат затем усреднялся для получения более точной оценки. Для получения результатов использовался метод библиотеки time под название clock, который возвращает процессорное время. Процессор: AMD Ryzen 3 2700, 8 ядер, 16 потоков

Замер тиков работы алгоритмов при четных размерах матриц:

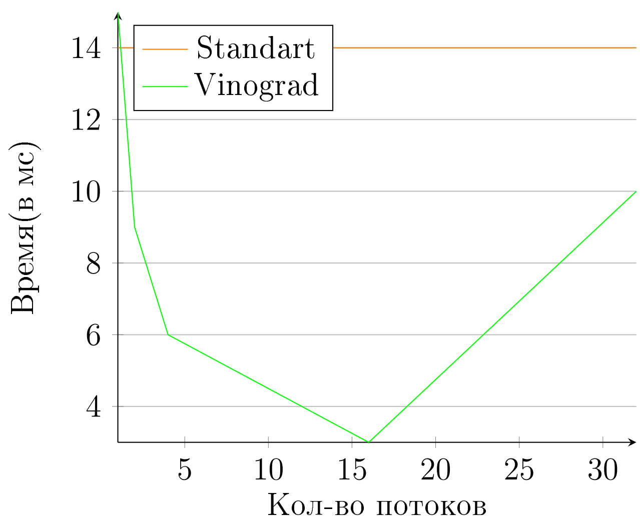

Рис. 4.1: График времени работы алгоритмов

4.2 Вывод

Из полученных результатов можно сделать вывод, что многопоточная реализация алгоритма Винограда умножения матриц эффективнее стандартной реализации.

Заключение

В результате проделанной работы были реализованы:

1. Однопоточный алгоритм Винограда.

2. Многопоточный алгоритм Винограда.

Эксперементально было подтверждено различие во временной эффективности однопоточных и многопоточных алгоритмов. Цель работы достигнута, все задачи выполнены.

Список использованных источников

1. Stothers A.J. On the Complexity of Matrix [Электронный ресурс] // era.lib.ed.ac.uk: [сайт]. [2010]. URL: https://www.era.lib.ed.ac.uk/bitstream/handle/184 2/4734/Stothers2010.pdf

2. Williams V.V. Multiplying matrices [Электронный ресурс] // http:// theory.stanford.edu: [сайт]. [2014]. URL: http://theory.stanford.edu/ virgi/matrixmult-f.pdf