Отчет по лабораторной работе 5
По предмету “Анализ алгоритмов”
По теме “Умножение матриц”

Фирсова Дарья ИУ7-56

2018

Введение

В лабораторной работе изучаются алгоритмы умножения матриц. Рассмотрены алгоритмы: стандартный и улучшенный алгоритм Винограда. Вычисления для каждого алгоритма выполняются параллельно.

Цель лабораторной работы: анализ, реализация и сравнительный анализ времени работы алгоритмов для различных размеров исходных матриц и количества потоков.

1 Аналитеская часть

В данном разделе приведены алгоритмы умножения.

1.1 Описание алгоритмов

1.1.1 Стандартный алгоритм умножения

Имеем две матрицы A и B размерностями M x N и N x Q соответственно.
Тогда результирующей матрицей будет матрица C размером M x Q, где $c_{ij} = \sum_{r=1}^{n} a_{ir} \cdot b_{rj}, (i = 0, 1, 2...m, j = 0, 1, 2...q)$.

1.1.2 Алгоритм Винограда

Пусть i-я строка матрицы A - вектор $\vec{U}$, а j-й столбец матрицы B - вектор $\vec{V}$.

Тогда $C_{ij} = \boxed{u_1\ u_2\ u_3\ u_4} \cdot \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \end{bmatrix} = u_1v_1 + u_2v_2 + u_3v_3 + u_4v_4 =$

$(u_1+v_2)(u_2v_1) + (u_3+v_4)(u_4+v_3)$ - u_1u_2 - u_3u_4 - v_1v_2 - v_3v_4.

"Хвост"для $\vec{U}$ вычисляется заранее и используется повтороно при умножении на каждый столбец матрицы B. Аналогично для вектора $\vec{V}$

Если вектора $\vec{U}$ и $\vec{V}$ нечетной длины, то к приведенным выше вычислениям, добавляем C_{ij} += $U_{N-1} \cdot V_{N-1}, \forall i, j$

2 Конструкторская часть

В данном разделе представлены схемы алгоритмов

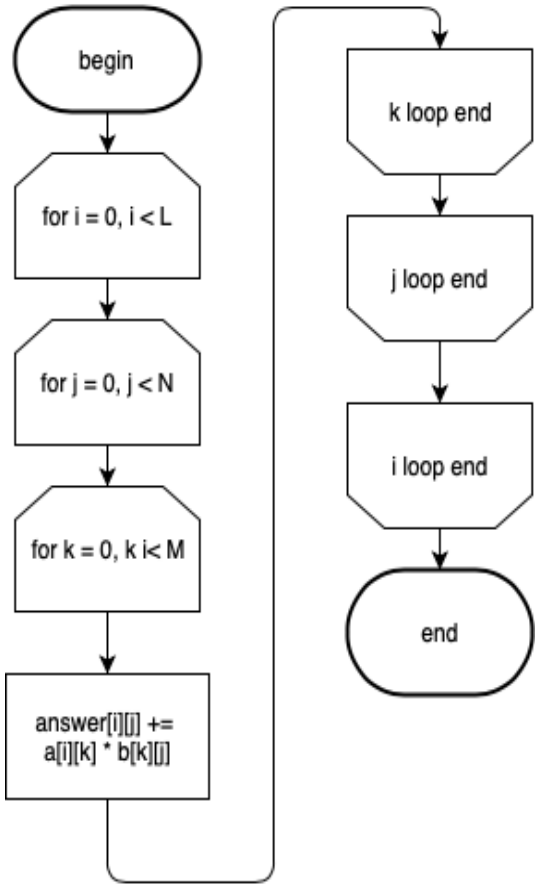


Рис. 1: Схема стандартного алгоритма

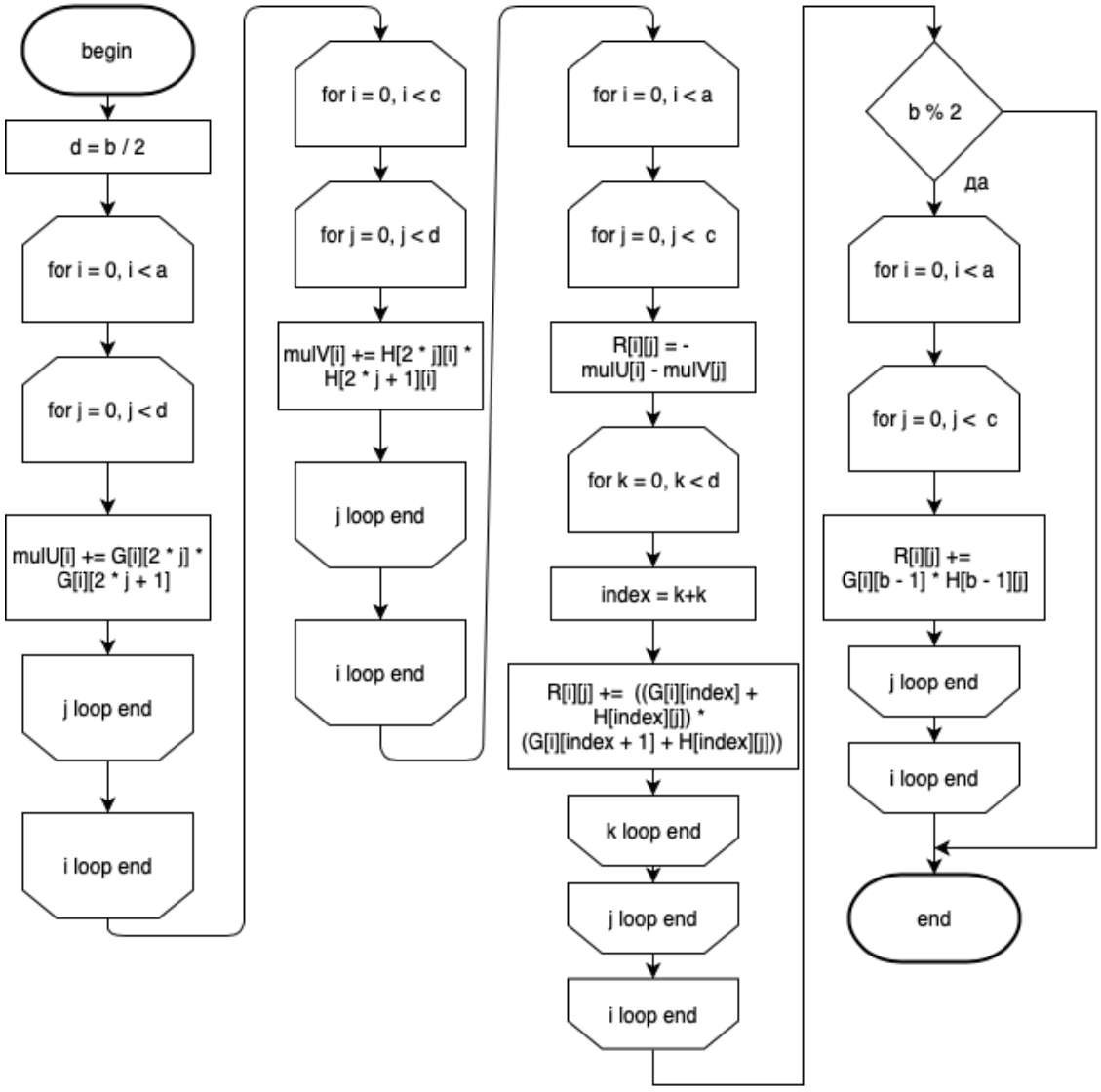


Рис. 2: Схема алгоритма улучшенного Винограда

3 Технологическая часть

В этом разделе приведена реализация функций, указан язык программирования и необходимые модули.

3.1 Средства реализации

В данной работе использовался язык Python 3.6, в среде Pycharm. Для измерения времени использовался модуль time, измерения производились в секундах.

3.2 Листинг кода

```
from random import randint
import time
import threading
import numpy as np


def multiply(a, b, c, start, fin):
    n = len(a[0])
    m = len(b[0])
    for i in range(start, fin):
        for j in range(m):
            for k in range(n):
                c[i][j] = c[i][j] + a[i][k]*b[k][j]


def winograd(a, b, c, start, fin):
    m = len(a)
    n = len(a[0])
    t = n//2+1
    row_fact = [0 for x in range(m)]
    column_fact = [0 for x in range(len(b[0]))]

    for i in range(m):
        row_fact[i] = 0
        for j in range(1, t):
            row_fact[i] = row_fact[i] + a[i][2*j-2] * a[i][2*j-1]

    for i in range(start, fin):
        column_fact[i] = 0
        for j in range(1, t):
            column_fact[i] = column_fact[i] + b[2*j-2][i] * b[2*j-1][i]

    for i in range(m):
        for j in range(start, fin):
            c[i][j] = -row_fact[i] - column_fact[j]
            for k in range(1, t):
                c[i][j] = c[i][j] + (a[i][2*k-2]+b[2*k-1][j])*(a[i][2*k-1] + b[2*k-2][j])

    if (n % 2 == 1):
        for i in range(m):
            for j in range(start, fin):
                c[i][j] = c[i][j] + a[i][n-1]*b[n-1][j]

```

```python

if __name__ == '__main__':
    f1 = open('multest.txt', 'w')
    f2 = open('winotest.txt', 'w')
    for leng in range(100, 807, 100):
        a = [[randint(0, 10) for i in range(leng)] for j in range(leng)]
        b = [[randint(0, 10) for i in range(leng)] for j in range(leng)]
        r = np.matmul(a,b)
        for threadn in [1,2,4,8]:
            timemul = []
            timewino = []
            for j in range(1):

                threads = []
                c = [[0 for p in range(leng)] for d in range(leng)]
                shag = leng / threadn
                end = shag
                start = 0
                for i in range(threadn):
                    threads.append(threading.Thread(target=multiply, args=(a, b, c, int(
    start), int(end))))
                    end += shag
                    start += shag
                tb = time.time()
                for thread in threads:
                    thread.start()
                for thread in threads:
                    thread.join()
                timemul.append(time.time() - tb)


                threads = []
                c = [[0 for i in range(leng)] for j in range(leng)]
                shag = leng / threadn
                end = shag
                start = 0
                for i in range(threadn):
                    threads.append(threading.Thread(target=winograd, args=(a, b, c, int(
    start), int(end))))
                    end += shag
                    start += shag
                tb = time.time()
                for thread in threads:
                    thread.start()
                for thread in threads:
                    thread.join()
                timewino.append(time.time() - tb)

            averagemul = sum(timemul)
            averagewino = sum(timewino)

            print('[illegible] {} [illegible] {} [illegible]
    {}'.format(leng, threadn, averagemul))
            print('[illegible] {} [illegible] {} [illegible] {}'
    .format(leng, threadn, averagewino))
            f1.write("{},{},{}\n".format(leng, threadn, averagemul))
```

```
96                f2.write("{},{},{}\n".format(leng, threadn, averagewino))
```

Листинг 1. Реализация алгоритмов.

4 Экспериментальная часть

В данном разделе будут приведены примеры работы алгоритмов и произведены замеры времени. Тестирование производилось на компьютере с процессором Intel Core i5 (I5-6267U) и оперативной памятью 8 Гб.

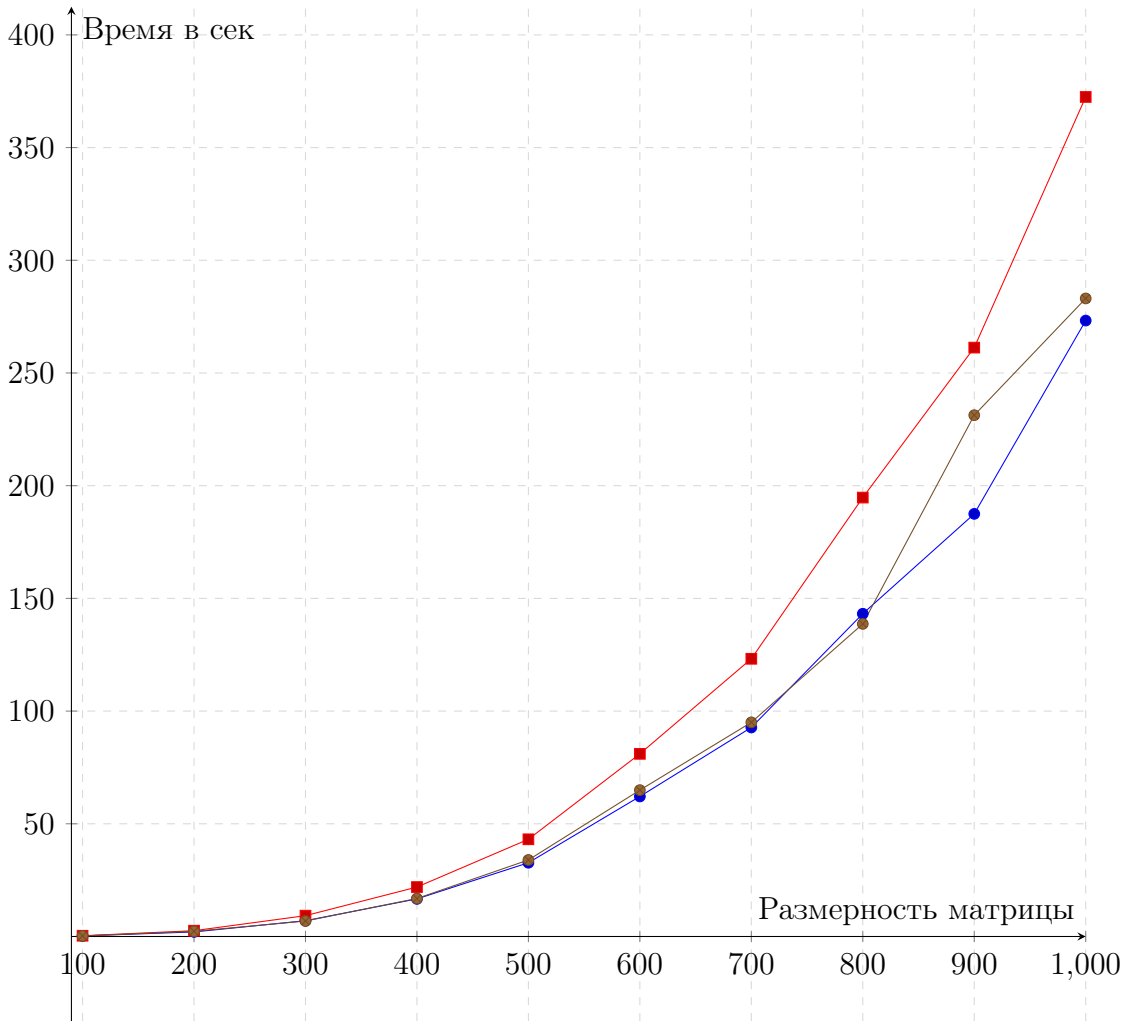
4.1 Примеры работы

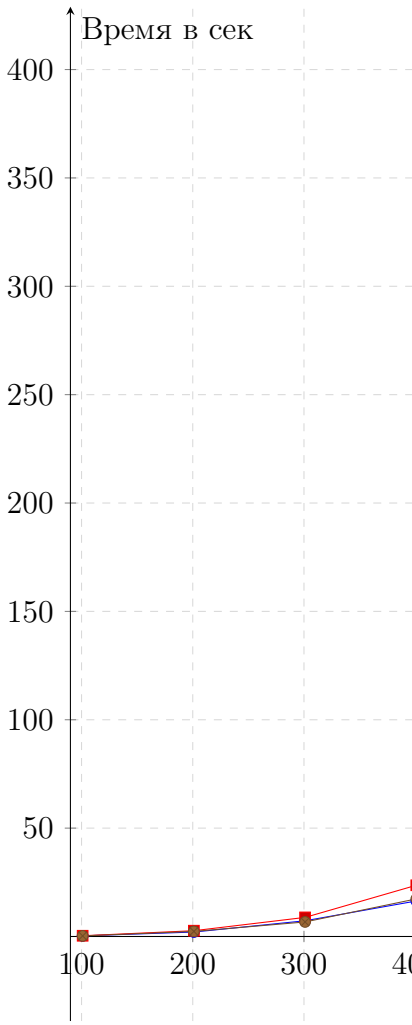
Пример результата работы умножения матриц. Так как в данной реализации генерируются случайные значения, то для проверки результата использовалась библиотека Numpy. При одинаковых входных данных алгоритмы выдают одинаковый результат, который сравнивается с результатом умножния с помощью функции numpy.matmul(). Для вычисления используются квадратные матрицы.

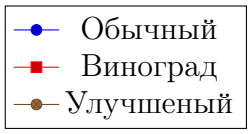
$$\begin{bmatrix} 12 & 14 & 20 \\ 24 & 14 & 16 \end{bmatrix} + \begin{bmatrix} 11 & 15 \\ 8 & 15 \\ 14 & 15 \end{bmatrix} = \begin{bmatrix} 524 & 690 \\ 600 & 810 \end{bmatrix}$$

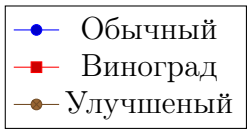
4.2 Сравнительный анализ

Сравнение алгоритмов стандартного умножения и улучшенного алгоритма Винограда в зависимости от количества потоков. На графиках приведены замеры времени работы для матрицы. Каждый эксперимент проводился 30 раз, результат - среднее арифметическое замеров времени.


Время в сек
400
350
300
250
200
150
100
50
Размерность матрицы
100
200
300
400
500
600
700
800
900
1,000


Время в сек
400
350
300
250
200
150
100
50
100
200
300
40


Обычный
Виноград
Улучшеный


Обычный
Виноград
Улучшеный

4.3 Вывод

5 Заключение

В лабораторной работе разработаны алгоритмы параллельного вычисления для умножения матриц. Разработаны программы по этим алгоритмам, проведены тесты по времени, произведен сравнительный анализ алгоритмов. Для составления отчета использован Latex.