Отчет по лабораторной работе 6
По предмету “Анализ алгоритмов”
По теме “Параллельные вычисления”

Фирсова Дарья ИУ7-56

2019

Введение

В лабораторной работе изучается алгоритм конвейера для параллельных вычислений. Используется работа с разделяемыми переменными и средствами синхронизации.
Цель лабораторной работы: разработка алгоритма параллельных вычислений с использованием 3-х очередей, мьютексов и написание логов.

1 Аналитеская часть

В данном разделе приведены алгоритмы умножения.

1.1 Описание алгоритмов

Конвейер состоит из трех уровней и трех очередей. На первом уровне элемент поступает из первой очереди, и создаются два других числа возведением исходного в 2 и 4 степень. Затем оба элемента поступают во вторую очередь. На втором уровне элемент забирается элемент из второй очереди, возводится в степень и поступает в 3 очередь. На третьем уровне к полученному элементу прибавляется число и "лента"завершает работу.

2 Конструкторская часть

В данном разделе представлены схемы алгоритмов

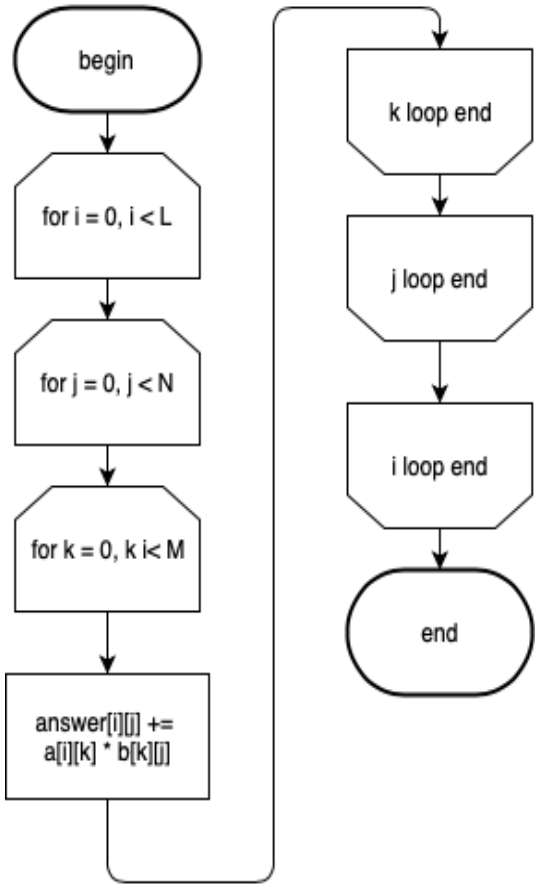


Рис. 1: Схема стандартного алгоритма

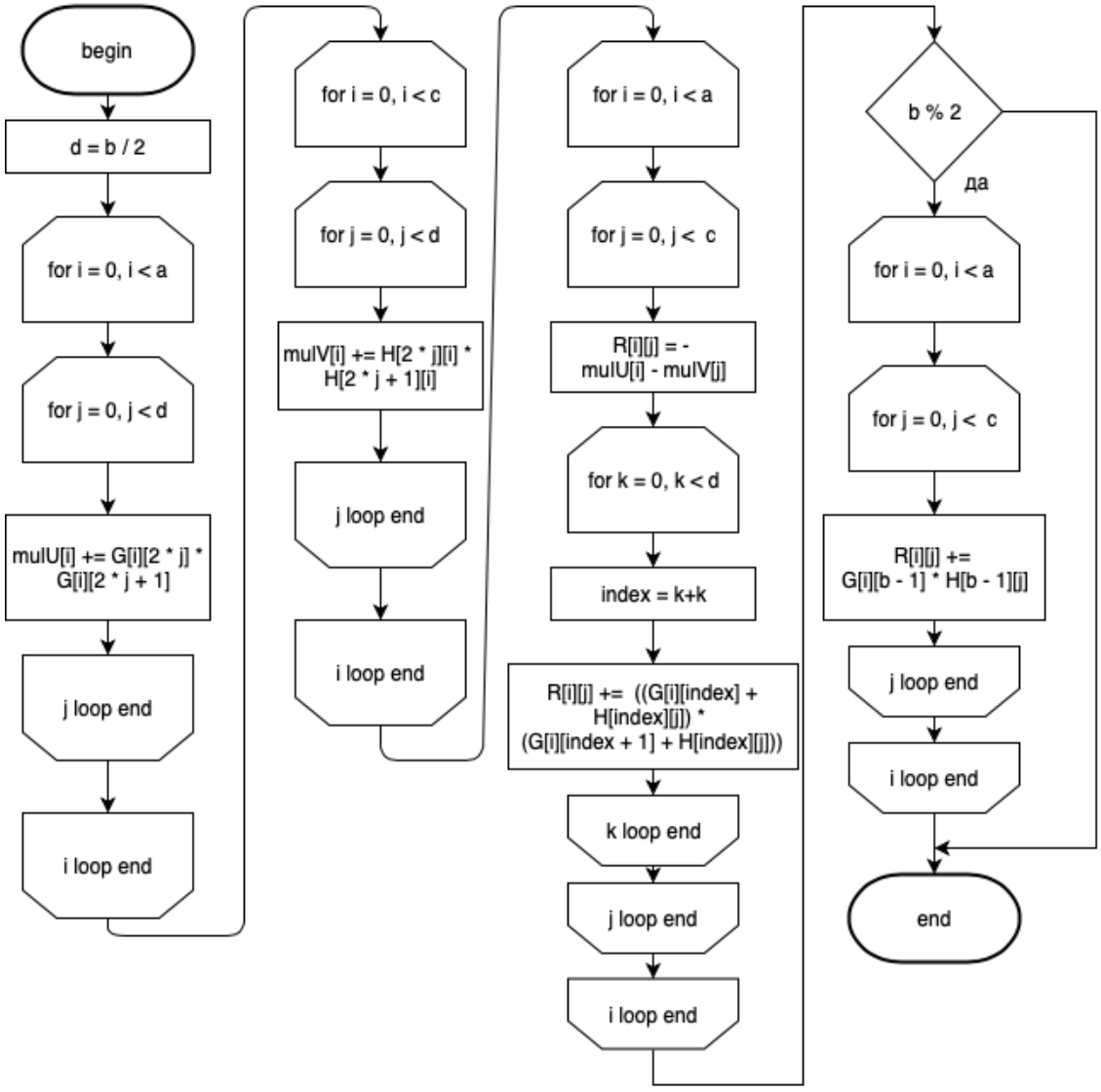


Рис. 2: Схема алгоритма улучшенного Винограда

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. Для реализации потоков используется threading, функция Thread которого позволяет создавать потоки. Функция получает на вход имя функции, которая будет реализована в потоке и ее аргументы.

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]


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))
```

```
[U+FFFD] [U+FFFD] print([U+FFFD][U+FFFD][U+FFFD][U+FFFD][U+FFFD][U+FFFD][U+FFFD][U+FFFD][U+FFFD]{}[U+FFFD][U+FFFD][U+FFFD][U+FFFD]{}[U+FFFD][U+FFFD][U+FFFD]{}[U+FFFD] [U+FF
    .format(leng, threadn, averagewino))
            f1.write("{},{},{}\n".format(leng, threadn, averagemul))
            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.

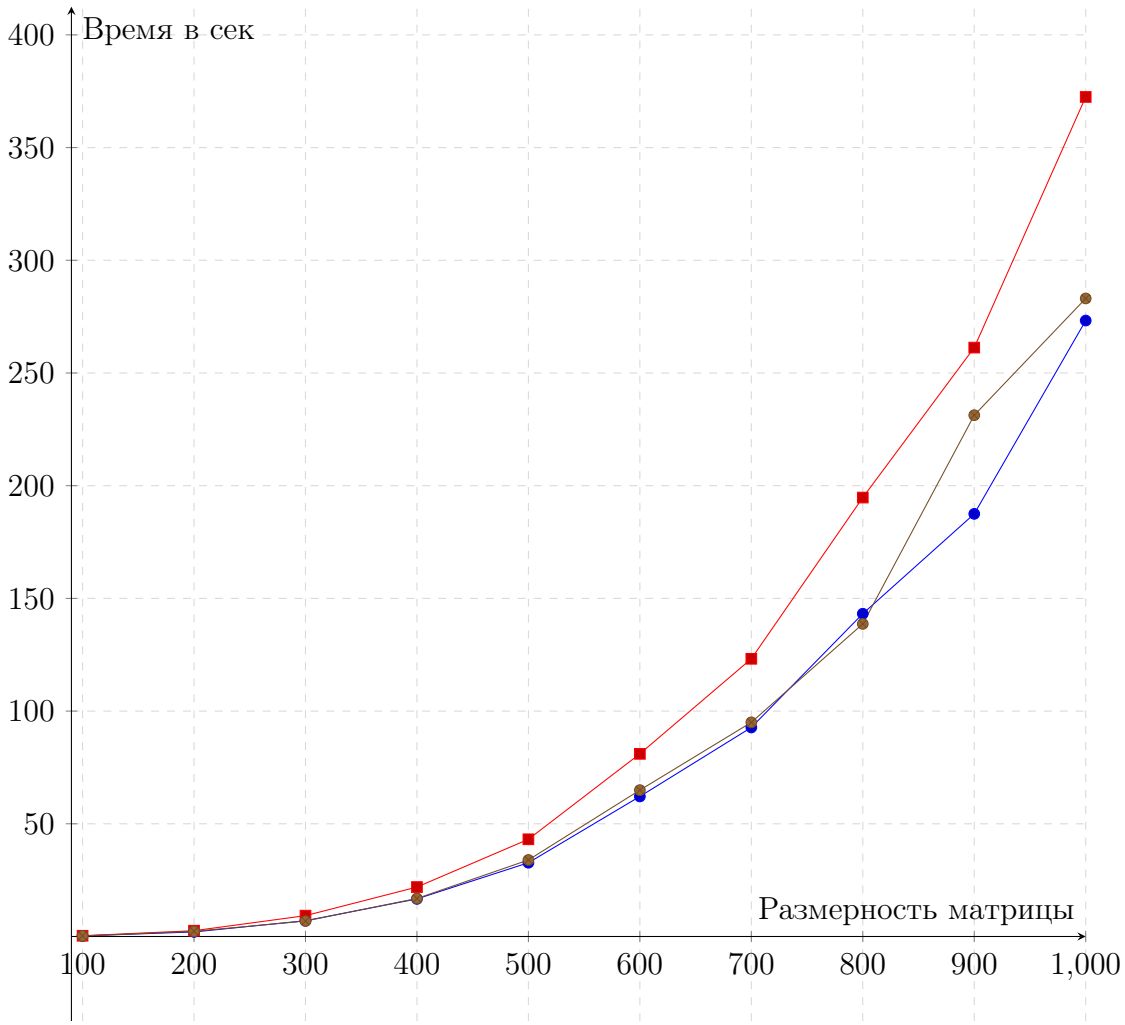
4.1 Примеры работы

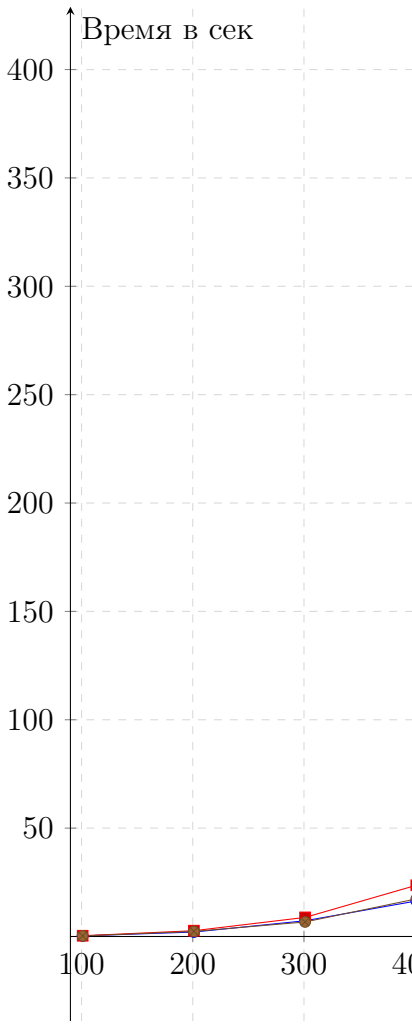
Пример результата работы умножения матриц. Так как в данной реализации генерируются случайные значения, то для проверки результата использовалась библиотека Numpy. При одинаковых входных данных алгоритмы выдают одинаковый результат, который сравнивается с результатом умножния с помощью функции numpy.matmul().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

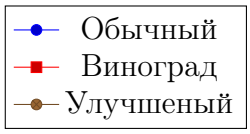
$$\begin{bmatrix} 12 & 14 & 20 \\ 24 & 14 & 16 \end{bmatrix} + \begin{bmatrix} 11 & 15 \\ 8 & 15 \\ 14 & 15 \end{bmatrix} = \begin{bmatrix} 524 & 690 \\ 600 & 810 \end{bmatrix}$$

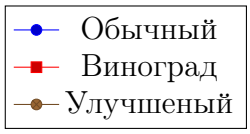
4.2 Сравнительный анализ

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.