**Федеральное государственное бюджетное образовательное учреждение   
высшего образования**

**«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»**

**(МГТУ им. Н.Э. Баумана)**

**Анализ алгоритмов**

**Лабораторная работа №5**

**Отчёт на тему:**

# «Параллельное умножение матриц»

Выполнила:

Янова Даниэлла

ИУ7-53

**Москва, 2018**

**Введение**

**Цель работы:** изучение алгоритмов умножения матриц. В данной лабораторной работе рассматривается распараллеленная реализация стандартного алгоритма умножения матриц, алгоритм Винограда.

1. **Аналитическая часть**

В данном разделе приведены алгоритмы умножения матриц.

**1.1 Умножение матриц. Стандартный алгоритм**

Матрицей называют математический объект, эквивалентный двумерному массиву. Матрица является прямоугрльной (в частных случаях – квадратной) таблицей, совокупностью строк и столбцов, на пересечении которых находятся элементы матрицы. Количество строк и столбцов является размерностью матриц. Для матриц определена операция умножения.

Пусть даны две матрицы – матрица А размером m на n и матрица B размером n на l.

A =, B = , C =

При этом

Стандартный алгоритм умножений матриц А и В выполняет m\*n\*l умножений и m\*(n-1)\*l сложений.

**1.2 Алгоритм Винограда.**

Очевидно, что каждый cij в результирующей матрице является скалярным произведением соответствующих столбца и строки. Такое умножение допускает предварительное вычисление части результата.

Имеются два вектора – V = (v1, v2, v3, v4) и W = (w1, w2, w3, w4).

«Стандартное» перемножение:

Перемножение по Винограду:

Данный алгоритм позволяет выполнить предварительную обработку матрицы и запомнить значения для каждой строки/столбца матриц.

* 1. **Распараллеливание задачи на CPU**

В рамках данной лабораторной работы производилось распараллеливание задачи по потокам. В CPU для данной цели используются treads.

CPU – central processing unit – это универсальный процессор, также именуемый процессором общего назначения. Он оптимизирован для достижения высокой производительности единственного потока команд. Доступ к памяти с данными и инструкциями происходит преимущественно случайным образом.

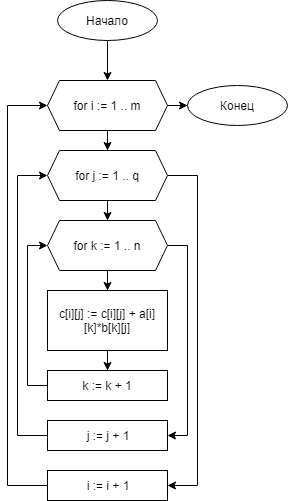
Для того, чтобы повысить производительность CPU ещё больше, они проектируются специально таким образом, чтобы выполнять как можно больше инструкций параллельно. Например, для этого в ядрах процессора используется блок внеочередного выполнения команд.

Но, несмотря на это, CPU всё равно не в состоянии осуществить параллельное выполнение большого числа инструкций, так как расходы на распараллеливание инструкций внутри ядра оказываются очень существенными. Именно поэтому процессоры общего назначения имеют не очень большое количество исполнительных блоков.

2. **Конструкторская часть**

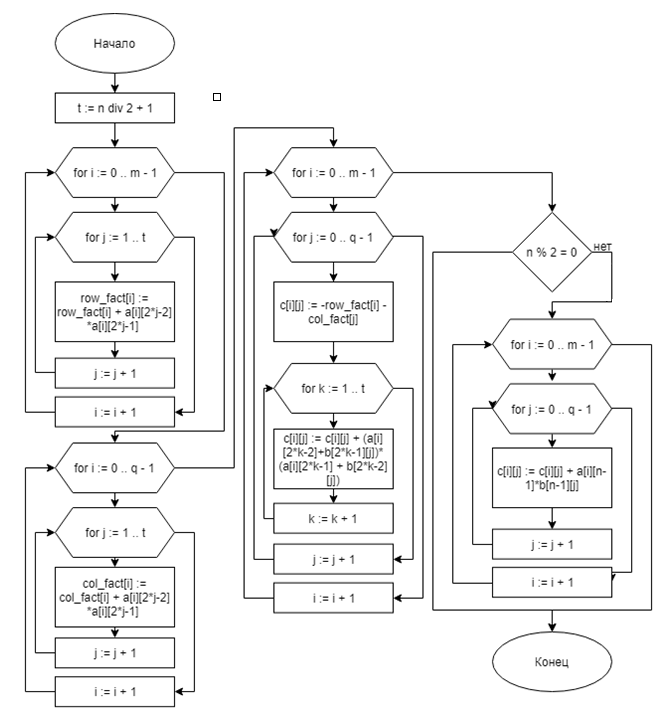
В данном разделе представлены схемы алгоритмов.

* 1. **Стандартный алгоритм**

****

**Рисунок 1.1 – Схема стандартного алгоритма**

* 1. **Алгоритм Винограда**



**Рисунок 1.2 – Схема алгоритма Винограда**

**3. Технологическая часть**

В данном разделе представлена реализация алгоритмов, указан язык программирования, а также необходимые модули.

**3.1 Средства реализации**

Для выполнения данной лабораторной работы использовался язык Python 3.7.1 в среде Pycharm. Замены времени проводились с использованием функции process\_time\_ns, входящей в библиотеку time Python версии 3.7.

**3.2 Листинг кода**

**import threading**

**from** **math** **import** ceil

**def** standart\_matrix\_mult(a, b, c, k1, k2):

n = len(a[0])

q = len(b[0])

**for** i **in** range(k1, k2):

**for** j **in** range(q):

**for** k **in** range(n):

c[i][j] = c[i][j] + a[i][k]\*b[k][j]

**def** classic\_vinograd\_mult(a, b, c, k1, k2):

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(k1, k2):

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(k1, k2):

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(k1, k2):

c[i][j] = c[i][j] + a[i][n-1]\*b[n-1][j]

**def** parallel(a, b, func, num):

n = len(a)

c = [[0 **for** x **in** range(n)] **for** y **in** range(n)]

k = n / num

tmp = k

pr\_tmp = 0

threads = []

**for** i **in** range(num-1):

threads.append(threading.Thread(target=func,

args=(a, b, c, int(pr\_tmp),int(tmp))))

tmp+=k

pr\_tmp+=k

threads.append(threading.Thread(target=func,

args=(a, b, c, int(pr\_tmp), ceil(tmp))))

**for** i **in** threads:

i.start()

**for** i **in** threads:

i.join()

**return** c

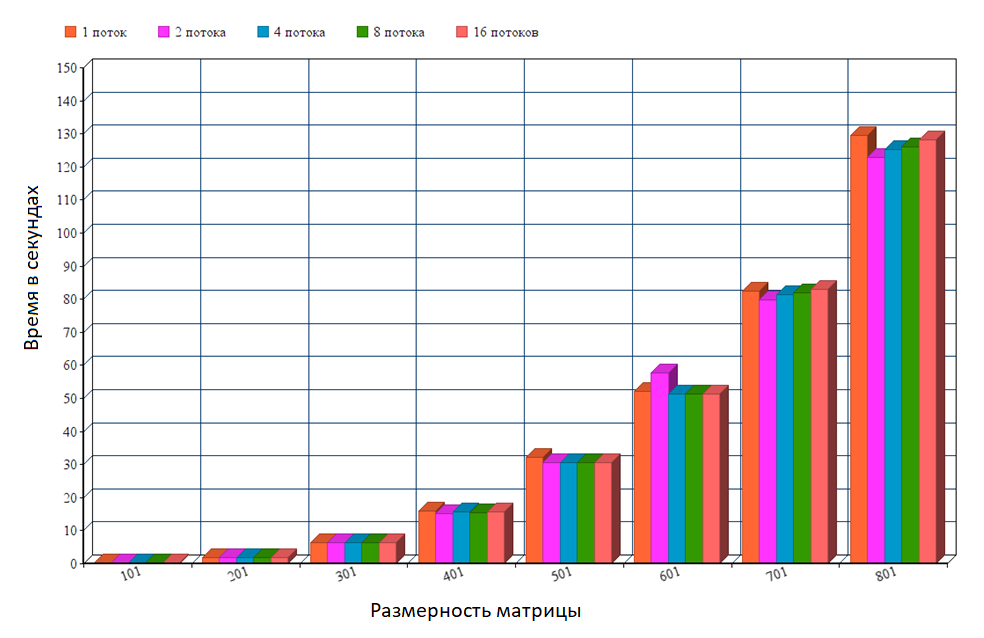
Листинг 1.1 – Реализация параллельности

1. **Экспериментальная часть**

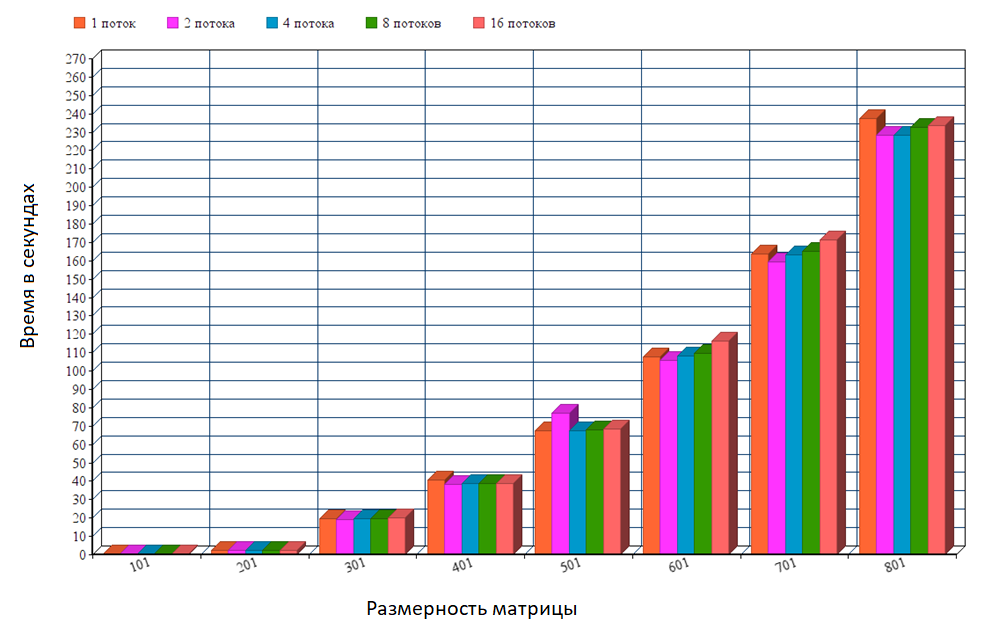
В данном разделе будут произведены замеры времени работы алгоритмов. Для исследования скоростных характеристик был использован компьютер на базе процессора Intel Core i7-6600U, содержащий 16 гигабайт оперативной памяти. Модуль тестирования запускался с жестокого диска под операционной системой Windows. Жесткий диск имел среднюю скорость передачи данных при чтении 2629 Мбайт/с и 798 Мбайт/с при записи.

**4.1 Тестирование алгоритмов**

Были проведены исследования зависимости времени работы алгоритмов от количества потоков. Замеры времени проводились для 1, 2, 4, 8 и 16 потоков. Временные замеры проводятся путём многократного проведения эксперимента и деления результирующего времени на количество итераций эксперимента. Замены времени проводились с использованием функции process\_time\_ns, входящей в библиотеку time Python версии 3.7



**Рисунок 4.1 – Замеры времени для классического алгоритма**



**Рисунок 4.2 – Замеры времени для алгоритма Винограда**

**Заключение**

В данной работе были исследованы алгоритмы умножения матриц и их распараллеленые реализации. Для параллельной реализации каждого алгоритма были проведены тесты, также был произведен сравнительный анализ времени работы алгоритмов в зависимости от количества потоков.