Отчет по лабораторной работе №3
по курсу "Анализ алгоритмов"
по теме "Изучение алгоритмов сортировки"

Студент: Барсуков Н.М. ИУ7-56
Преподаватель: Волкова Л.Л., Строганов Ю.В.

Содержание

Введение

Умножение матриц — это один из базовых алгоритмов, который широко применяется в различных численных методах, и в частности в алгоритмах машинного обучения. Многие реализации прямого и обратного распространения сигнала в сверточных слоях нейронной сети базируются на этой операции. Так порой до 90-95% всего времени, затрачиваемого на машинное обучение, приходится именно на эту операцию. Так же это один из немногих алгоритмов, который позволяет эффективно задействовать все вычислительные ресурсы современных процессоров и графических ускорителей. Поэтому не удивительно, что многие алгоритмы стараются свести к матричному умножению — дополнительная расходы, связанные с подготовкой данных, как правило с лихвой окупаются общим ускорением алгоритмов. Перемножения матриц также используются в прикладной физике, математике, математической статистике и многих других прикладных науках

1 Аналитический раздел

В этом разделе описаны задачи, необходимые для достижения цели. Приведено математическое и детальное описание трех алгоритмов умножения матриц:

1) классический;
2) винограда;
3) винограда Модифицированный.

1.1 Постановка задачи

Цель: изучение способа распараллеливания алгоритма Винограда. В данной работе рассматривается распараллеленная реализация стандартного алгоритма умножения матриц, алгоритма Винограда.

Задачи:

1) рассмотреть задачу умножения матриц;
2) произвести анализ сложности алгоритмов умножения матриц;
3) запрограммировать алгоритмы и сделать их работу параллельной;
4) сделать замеры времени для алгоритмов;
5) результаты экспериментов сравнить с теоретическими оценками трудоёмкости;
6) сделать выводы.

1.2 Классический подход

Предположим, что необходимо получить матрицу $C_{(a,c)} = A_{(a,b)} * B_{(b,c)}$. Для нахождения значений элементов матрицы $C_{(a,c)}$ используют следующие выражение[**makkonell**]

$$C_{i,j} = \sum_{K}(A_{i,k}B_{i,k}) \tag{1}$$

Классический алгоритм напрямую реализует эту формулу

1.3 Алгоритм Винограда

Можно заметить, что элементы из суммы выражения 1 можно переписать как:

$$A_{i,k-1}B_{k-1,j}+A_{i,k}B_{k,j} = (A_{i,k-1}+B_{k,j})(A_{i,k}+B_{k-1,j})-A_{i,k-1}A_{i,k}-B_{k-1}B_{k,j} \quad (2)$$

т. е. как сумму произведения сумм и двух произведений. Учитывая, что упомянутые два произведения можно рассчитать заранее для обработки двух элементов матрицы теперь нужно не сложение и два умножения, а умножение и два сложения, что проще с точки зрения вычислений. Таким образом, алгоритм Винограда состоит в следующем:

1. совершить расчет заранее двух произведений для каждого ряда и столбца матрицы-результата (одно произведение считается для ряда, другое для столбца). Для хранения результатов используется промежуточный буфер;

2. по вышеприведённой формуле осуществить расчёт каждого элемента матрицы;

3. в случае, если в произведении $A_{(a,b)XB_{b,c}}$ b – нечётное число, пройтись во второй раз по матрице, дополняя элементы $C_{i,j}$ недостающим элементом (который не был описан вышеописанной суммой). Можно заметить, что пункт 3 необходимо выполнять только в некоторых случаях, но если это происходит, то получается существенное увеличение времени работы алгоритма.

1.4 Оптимизированный алгоритм Винограда

1. Внутри тройного цикла накапливать результат в буфер, а вне цикла сбрасывать его в ячейку матрицы.

2. Заменить MulH[i] = MulH[i] + ... на MulH[i] += ... (аналогично для MulV), где MulH и MulB – временные массивы для предварительного рассчета сумм произведений

1.5 Распараллеливание задачи на CPU

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

Central Processing Unit (CPU) – это универсальный процессор, также именуемый процессором общего назначения. Доступ к памяти с данными и инструкциями происходит преимущественно случайным образом.

Архитектура x86 проектировалась 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 инструкции внутри ядра увеличивают время работы программы. Именно поэтому процессоры общего назначения имеют не очень большое количество исполнительных блоков.

1.6 Вывод

В данном разделе детально описаны 3 алгоритма умножения матриц: Классический, Винограда, модифицированная версия Винограда. Кратко описано про распараллеливание задач.

2 Конструкторский раздел

В данном разделе введена модель трудоемкости алгоритмов. Посчитана сложность. Приведены схемы алгоритмов

2.0.1 Оценка трудоемкости алгоритмов

Используется С-подобная модель оценки трудоемкости.

Трудоемкость операций:

1) трудоемкость операций: +, -, =, + =, - =, <, > ==, ++ равна 1;

2) трудоемкость операций: *, /, % равна 2;

3) трудоемкость операции доступа к элементу памяти: [...] равна 3.

Трудоемкость смены ячеек памяти местами будем считать 9, так как производится 3 обращения к памяти.

Цикл будет оцениваться о фактически выполненным операциям из перечня выше.

Условный оператор if будет фактически оценен как сумма стоимости операций в условии и трудоемкости различных ветвей (в лучшем случае и в худшем случае). Стоимость условного перехода из уловия в одну из ветвей решения полагается равной 0.

2.0.2 Классический алгоритм умножения матриц

Умножаются 2 матрицы M1(M, N), M2(N, Q). Ниже приведена формула рассчета сложности в в соответствии с моделью оценки трудоемкости.

Итоговая формула:

$$\boxed{O(M,N,Q) = 8 * M * N * Q + 4 * M * N + 4 * M + 2} \tag{3}$$

2.0.3 Алгоритм Винограда

Умножаются 2 матрицы M1(M, N), M2(N, Q)

1) цикл 1:

$$\boxed{2 + M(2 + 3 + \frac{N}{2} * (3 + 10))} \tag{4}$$

2) цикл 2:

$$\boxed{2 + Q(2 + 3 + \frac{N}{2} * (3 + 10))} \tag{5}$$

3) цикл 3:

$$\boxed{2 + M(2 + 2 + Q(2 + 7 + 3 + \frac{N}{2} * (3 + 20)))} \tag{6}$$

Условие четности/нечетности:

$$\boxed{1 + \begin{cases} 2 + M(2 + 2 + Q(2 + 10)), \textbf{N нечетное (худший вариант)} \\ 0, \textbf{ иначе (лучший вариант)} \end{cases}} \quad (7)$$

Итоговая формула получается из суммы 4, 5, 6, 7

2.0.4 Оптимизированный алгоритм Винограда

Умножаются 2 матрицы M1(M, N), M2(N, Q)

1) цикл 1:

$$\boxed{2 + M(2 + 2 + d(2 + 10))} \quad (8)$$

2) цикл 2:

$$\boxed{2 + Q(2 + 2 + d(2 + 10))} \quad (9)$$

3) цикл 3:

$$\boxed{2 + M(2 + 2 + Q(2 + 5 + 2 + d(2 + 20)))} \quad (10)$$

Условие четности/нечетности:

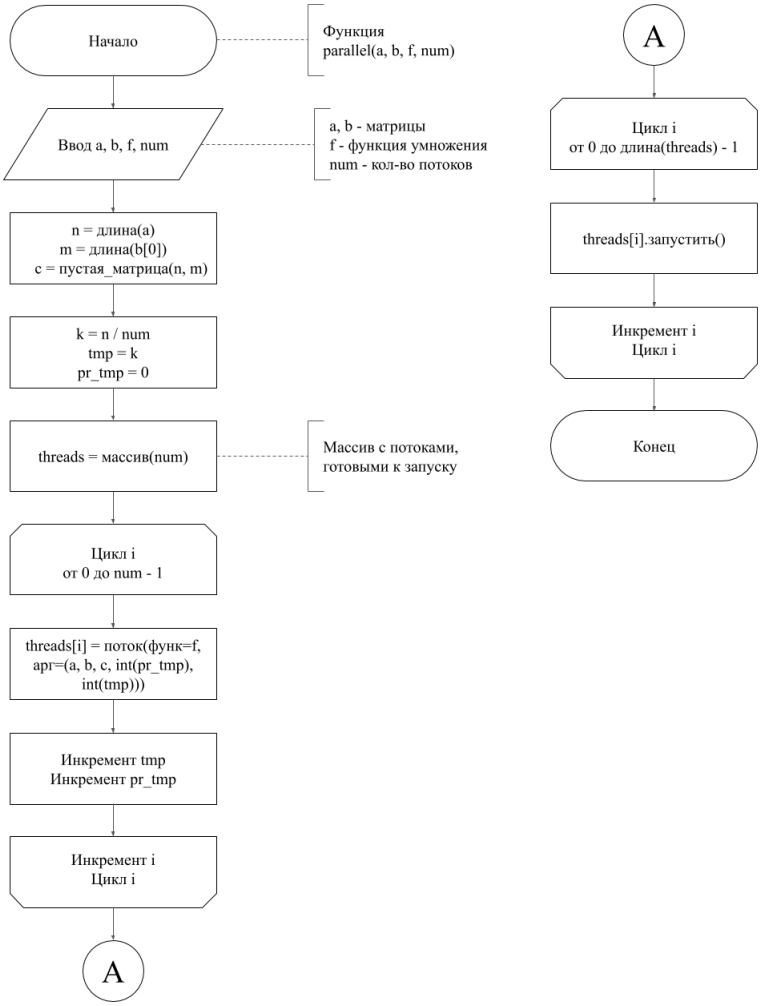
$$\boxed{1 + \begin{cases} 2 + M(2 + 2 + Q(2 + 10)), \textbf{N нечетное (худший вариант)} \\ 0, \textbf{ иначе (лучший вариант)} \end{cases}} \quad (11)$$

Итоговая формула получается из суммы 8, 9, 10, 11

2.1 Схемы

В данном подразделе приведены следующие схемы:

1) Схема распараллеливания 1;
2) Распараллеленный алгоритм Винограда 2, 3.


Начало
Функция
parallel(a, b, f, num)
Ввод a, b, f, num
a, b - матрицы
f - функция умножения
num - кол-во потоков
n = длина(a)
m = длина(b[0])
c = пустая_матрица(n, m)
k = n / num
tmp = k
pr_tmp = 0
threads = массив(num)
Массив с потоками,
готовыми к запуску
Цикл i
от 0 до num - 1
threads[i] = поток(функ=f,
арг=(a, b, c, int(pr_tmp),
int(tmp)))
Инкремент tmp
Инкремент pr_tmp
Инкремент i
Цикл i
A
A
Цикл i
от 0 до длина(threads) - 1
threads[i].запустить()
Инкремент i
Цикл i
Конец

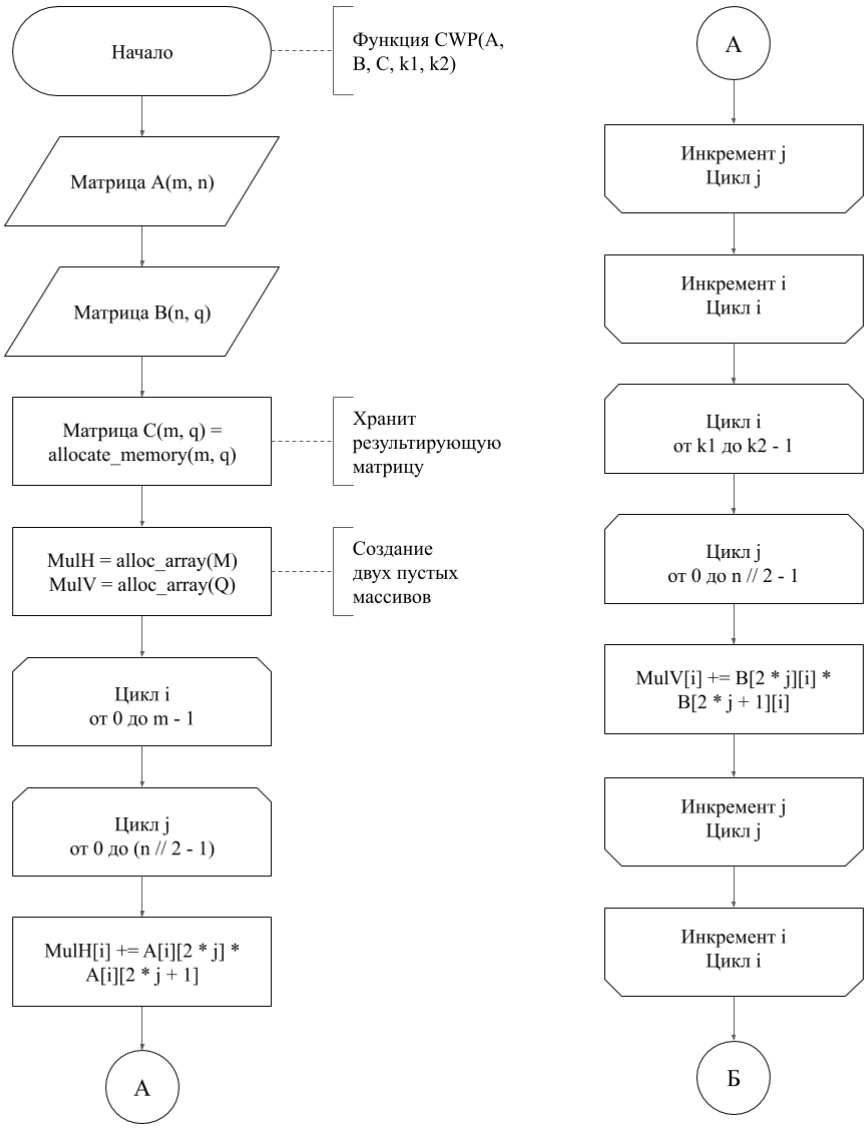


Рис. 2: Распараллеленный алгоритм Винограда. Часть 1

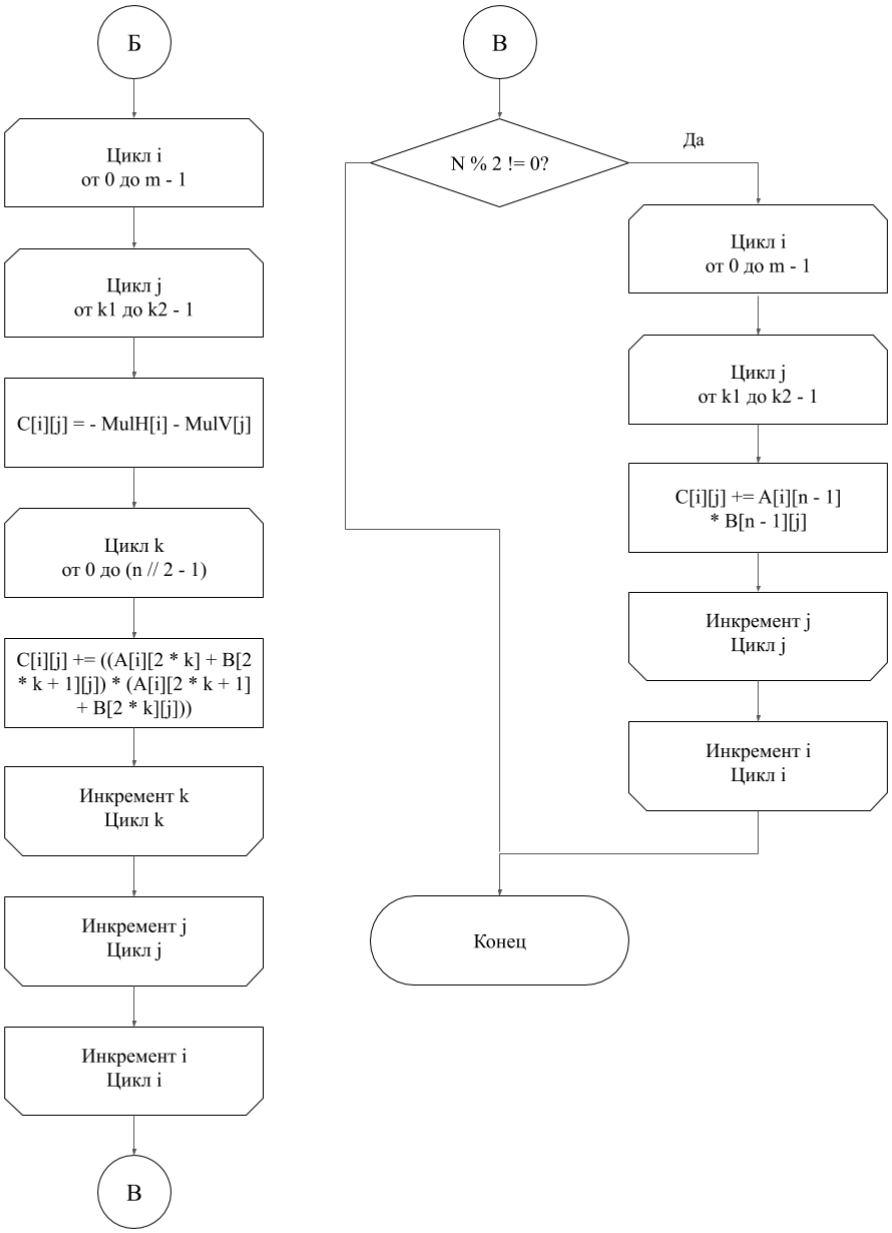


Рис. 3: Распараллеленный алгоритм Винограда. Часть 2

2.2 Вывод

В данном разделе была введена модель трудоемкости. Вычислена трудоемкость каждого алгоритма. Указаны схемы алгоритмов.

3 Технологический раздел

В данном разделе указаны минимальные системные требования. Описан используемый язык и среда разработки. Приведен листинг алгоритмов умножения матриц

3.1 Минимальные требования

Минимальные системные требования: PC с операционной системой Windows XP/Vista/7/8/10. Требуются устройства ввода: клавиатура, мышь. Устройство вывода: монитор.

3.2 Выбор языка и среды разработки

Для решения данной поставленной задачи, мной был выбран язык Python 3.8 по удобности и знания. Так же использовалась среда PyCharm 2019

3.3 Листинг

Листинг 1: Классический алгоритм умножения

```
def classic_matrix_mult(A, B):

    A, B = np.array(A), np.array(B)

    if len(B) != len(A[0]):
        print("Error! Different dimension!")
        return None

    n = len(A)
    m = len(A[0])
    t = len(B[0])

    C = np.zeros((A.shape[0], B.shape[1]))
    for i in range(n):
        for j in range(m):
            for k in range(t):
                C[i][k] += A[i][j] * B[j][k]
    return C
```

Листинг 2: Алгоритм Винограда с поддержкой параллельности

```
def classic_winograd_mult(A, B, C, k1, k2):
    M = len(A)
    N = len(B)
    Q = len(B[0])

    MulH = np.zeros(M)
```

```
    MulV = np.zeros(Q)

    for i in range(M):
        for j in range(N // 2):
            MulH[i] += A[i][2 * j] * A[i][2 * j + 1]

    for i in range(k1, k2):
        for j in range(N // 2):
            MulV[i] += B[2 * j][i] * B[2 * j + 1][i]

    for i in range(M):
        for j in range(k1, k2):
            C[i][j] = - MulH[i] - MulV[j]
            for k in range(N // 2):
                C[i][j] += (A[i][2 * k] + B[2 * k +
                    1][j]) * (A[i][2 * k + 1] + B[2 *
                    k][j])

    if N % 2:
        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]
```

Листинг 3: Функция распараллеливания вычислений алгоритма Винограда по столбцам

```
def parallel(a, b, f, num):
    n = len(a)
    m = len(b[0])
    c = np.zeros((n, m))

    k = n / num
    tmp = k
    pr_tmp = 0
    threads = []

    for i in range(num):
        threads.append(threading.Thread(target=f,
            args=(a, b, c, int(pr_tmp), int(tmp))))
        tmp += k
        pr_tmp += k

    for i in threads:
        i.start()

    for i in threads:
        i.join()

    return c
```

4 Исследовательский раздел

4.1 Характеристики оборудования

1. Компьютер:
   (a) Тип компьютера Компьютер с ACPI на базе x64;
   (b) Операционная система Microsoft Windows 10 Pro.
2. Системная плата:
   (a) тип ЦП DualCore Intel Core i5-6200U, 2700 MHz (27 x 100);
   (b) системная плата HP 8079;
   (c) чипсет системной платы Intel Sunrise Point-LP, Intel Skylake-U;
   (d) системная память 8072 МБ (DDR4 SDRAM).

Замеры времени проводились для матриц размера 100x100, 200x200, 300x300, 400x400, 500x500. Так как процессор, на котором выполнялись вычисления, имеет лишь 4 ядра, не было возможности произвести более обширное тестирование такое как, например, на GPU. Тем не менее, распараллеливание вычислений оказалось эффективнее классического алгоритма, даже учитывая небольшое число ядер процессора.

Очевидно, что производительность алгоритма на матрицах нечетного размера будет хуже, так как внутри алгоритма есть проверка на условие, после чего следует 2 вложенных цикла. Они будут причиной увеличения времени работы. Данное сравнение проводилось в лабораторной работе №2 и в данной работе производиться не будет.

Замер времени проводился с помощью библиотеки time в Python 3.8 и метода processtime(). На рисунке 1 видно, что во всех случаях попытка распараллелить алгоритм приводила к увеличению производительности алгоритма по времени и сокращению времени работы.

Распараллеливание проводилось с помощью библиотеки threading для Python 3.8. Как показал опыт, данная библиотека оказалось недостаточно эффективной в сравнении с распараллеливанием на других языках. В среднем получилось уменьшить время работа на 10% по сравнению с алгоритмом без распараллеливания.

На момент замера времени работало в среднем 76 активных процессов. На графике отображено, что при нескольких процессах работа алгоритма будет дольше, чем при одном. Это может зависеть от нескольких факторов таких, как: загруженность процессора другими процессами, обработка прерываний, ошибки внутри библиотеки для распараллеливания.

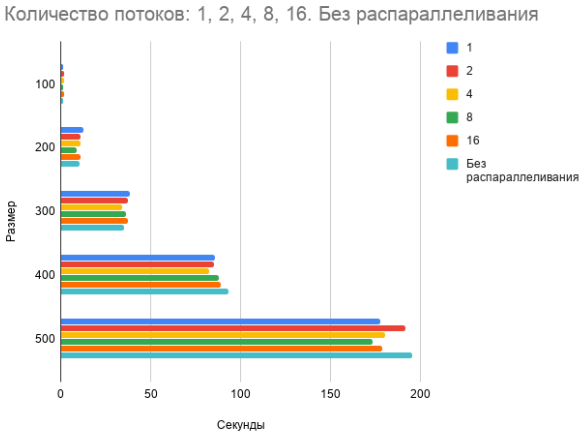


Рис. 4: Зависимость времени работы алгоритма Винограда от количества потоков

4.2 Вывод

В данном разделе указана характеристики оборудования на котором проводилось тестирование. Указаны графики сравнения работы алгоритма в зависимости от количества потоков. Подсчитано процентное соотношение скорости работы алгоритма в зависимости от количества потоков

5 Заключение

Опыт показал, не смотря, на проблемы с распараллеливанием на Pythone. Что рекомендуется применять распараллеливания алгоритма Винограда, умножения матриц. Для увеличения скорости работы.

Список литературы

[1] Дж. Макконнелл. Анализ алгоритмов. Активный обучающий подход.- М.:Техносфера, 2009.

[2] Электронный ресурс Python 3.8.1 documentation. Режим доступа https://docs.python.org/3/ - свободный. Последнее обращение (28.12.2019)