

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«ДОНСКОЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

**(ДГТУ)**

Факультет «Информатика и вычислительная техника»

Кафедра «Программное обеспечение вычислительной техники и автоматизированных систем»

**Лабораторная работа №4**

по дисциплине «Основы теории принятия решений»

**Выполнил:**

студент уч. гр. ВМО41

Волкова Эмилия Юрьевна

**Проверил:**

проф. Кобак В. Г.

Ростов-на-Дону

2025

**Введение**

Предметом области исследования распределения нагрузки и планирования задач является круг задач проектирования и организационного управления в многозадачных вычислительных системах, где необходимо найти оптимальное распределение ресурсов с учётом различных критериев. В таких системах задачи, представленные в виде матриц, требуют грамотного распределения между процессами, чтобы минимизировать нагрузку на систему и обеспечить эффективное выполнение вычислительных процессов.

Программирование для многопроцессорных вычислительных систем связано с распараллеливанием и синхронизацией вычислений, а также организацией выполнения параллельных вычислительных процессов. В рамках этого исследования рассматриваются различные алгоритмы распределения задач, целью которых является минимизация максимальной нагрузки на систему. Решение этой задачи важно для повышения производительности вычислительных систем и улучшения использования доступных ресурсов.

Модели параллельных программ и операционные характеристики процессов их выполнения лежат в основе построения расписаний, которое определяет порядок выполнения задач на вычислительных системах. С увеличением числа задач и процессоров сложность построения эффективных расписаний возрастает, что делает важным поиск оптимальных или приближенных решений. Алгоритмы, использующие различные критерии оптимизации, такие как квадратичный, кубический или минимаксный критерий, помогают справиться с этими вызовами и позволяют достигать решений, близких к оптимальным с точки зрения времени выполнения и минимизации нагрузки на систему.

**Постановка задачи**

В области теории расписаний задача оптимального распределения вычислительных заданий между многими процессами или устройствами является одной из ключевых для обеспечения эффективного функционирования многозадачных и многопроцессорных систем. При решении такой задачи важно учитывать различные критерии эффективности, такие как минимизация максимальной нагрузки на устройства или минимизация времени выполнения всех заданий. Однако, решение таких задач с помощью точных методов является сложным и непрактичным для больших систем, так как сложность решения возрастает экспоненциально с увеличением размерности задачи.

Одним из подходов, используемых для решения подобных задач, являются приближенные алгоритмы, основанные на эвристиках. В рамках данного исследования рассматривается алгоритм, предназначенный для распределения M заданий на N устройств с целью минимизации максимальной нагрузки на устройства, используя различные критерии оптимизации: квадратичный и кубический критерии, а также минимаксный критерий. Задания представлены в виде матрицы, где каждый элемент отражает время выполнения задания на соответствующем устройстве.

Рассматривается система из N устройств P = { p1, p2 , ... , pn}, каждое из которых выполняет задания. Множество заданий T = {t1, t2,...,tm} требует распределения по этим устройствам. Каждое задание ti может быть выполнено на любом устройстве pj, и время выполнения зависит от устройства и определяется матрицей Tτ={τ(ti, pj)}, где τ(ti,pj) — время выполнения задания ti на устройстве pj.

Цель задачи — найти такое распределение заданий, при котором максимальная нагрузка на устройства (время завершения работы устройства) будет минимальной. В качестве критериев оптимизации используются:

* Квадратичный критерий, где минимизируется сумма квадратов времени выполнения задач.
* Кубический критерий, который учитывает более сложные зависимости времени выполнения на каждом устройстве.
* Минимаксный критерий, направленный на минимизацию максимальной нагрузки на устройства, что позволяет сбалансировать распределение заданий.

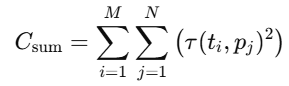
В результате работы программы алгоритмы сортировки и распределения выполняют расчеты для различных вариантов сортировки заданий (по возрастанию или убыванию их суммы по строкам матрицы) и для разных критериев оптимизации. Программа сравнивает результаты и помогает определить наиболее эффективный способ распределения заданий для данной системы с учетом выбранного критерия оптимальности.

### 1. ****Алгоритм для квадратичного и кубического критерия****

Пусть τ(ti,pj) — время выполнения задания ti на устройстве pj.

Требуется минимизировать суммарное время обработки заданий, при этом использован квадратичный критерий.

Оптимизация задачи для квадратичного критерия сводится к минимизации следующей функции:



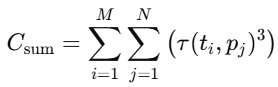
где:

* τ(ti,pj) — время выполнения задания ti на устройстве pj.
* Cуммируем квадраты всех времен выполнения заданий на соответствующих устройствах.

#### Шаги алгоритма:

1. **Инициализация:**
   * Создаем матрицу, которая будет хранить время выполнения для каждого задания на каждом устройстве.
   * Создаем вспомогательные массивы для отслеживания минимальных значений нагрузки и текущего состояния устройств.
2. **Распределение заданий:**
   * Для каждого задания ti и каждого устройства pj tit\_i выбираем наименьшее время, учитывая текущую нагрузку на устройства.
   * После выбора минимального времени для задания обновляем нагрузку устройства.
3. **Цель:**
   * Алгоритм находит распределение заданий по устройствам, которое минимизирует сумму квадратов времени выполнения.

Для алгорима **кубического критерия** аналогично:

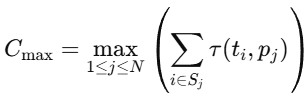


### 2. ****Алгоритм Плотникова–Зверева для минимаксного критерия****

Пусть τ(ti,pj) — время выполнения задания ti на устройстве pj.

Минимизируем **максимальное** время выполнения среди всех устройств, то есть минимизируем Cmax.

Оптимизация задачи для минимаксного критерия сводится к минимизации максимальной нагрузки на устройства:



где:

* Cmax — максимальное время завершения работы устройства pj.
* Sj — множество заданий, назначенных процессору pj.
* Мы минимизируем значение Cmax , что позволяет сбалансировать нагрузку на устройства.

#### Шаги алгоритма:

1. **Инициализация:**
   * Создается матрица с временем выполнения заданий на устройствах и массивы для отслеживания нагрузки на устройства.
2. **Распределение заданий:**
   * Для каждого задания выбирается устройство с наименьшей текущей нагрузкой.
   * Алгоритм пытается равномерно распределить задания между устройствами, чтобы минимизировать максимальное время выполнения на устройстве.
3. **Цель:**
   * Алгоритм Плотникова–Зверева находит распределение заданий, которое минимизирует максимальную нагрузку на любое устройство. Это соответствует минимизации максимального времени, которое потребуется для выполнения всех заданий.

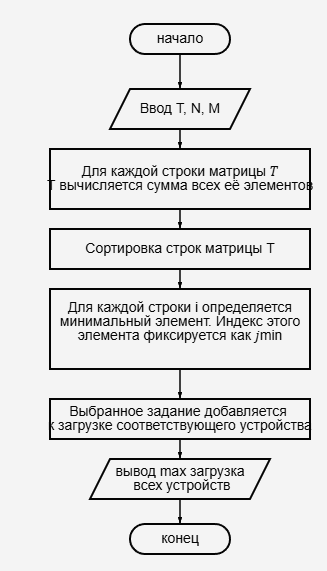


Рисунок 1 – Схема алгоритма алгоритма Плотникова-Зверева для минимаксного критерия

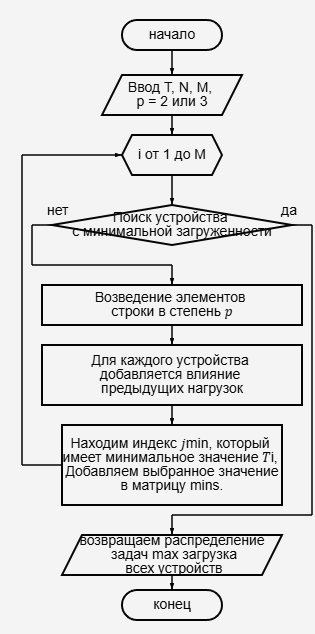


Рисунок 2 – Схема алгоритма для квадратичного и кубического критерия

**Листинг программы**

import random, numpy as np

import time

def sort(T, choice):

    match choice:

        case 0: return T

        case 1: return T[np.sum(T, axis=1).argsort()]

        case 2: return T[np.sum(T, axis=1).argsort()[::-1]]

def alg(T, N, M, p):

    matrix = T.copy()

    mins = np.zeros((M, N))

    lead = np.zeros(N)

    for row in range(0, M):

        if row != M-1:

            current\_row = matrix[row]

            Tt = np.power(current\_row, p)

            for i in range(0, len(lead)):

                Tt = Tt + lead[i]\*\*p

                Tt[i] -= lead[i]\*\*p

            ind = np.argmin(Tt)

            elem = current\_row[ind]

            mins[row][ind] = elem

            lead[ind] = elem

            matrix[row+1] = matrix[row+1] + lead

        elif row == M-1:

            current\_row = matrix[row]

            Tt = np.power(current\_row, p)

            for i in range(0, len(lead)):

                Tt = Tt + lead[i]\*\*p

                Tt[i] -= lead[i]\*\*p

            ind = np.argmin(Tt)

            elem = current\_row[ind]

            mins[row][ind] = elem

            lead[ind] = elem

    if p == 2:

        print('1', mins)

    elif p == 3:

        print('2', mins)

    tasks = np.array([])

    for i in np.transpose(mins): tasks = np.append(tasks, i[i != 0][-1])

    #print(tasks, max(tasks))

    return tasks, max(tasks)

def Plotnikov\_Zverev(matrix, N, M):

    start = time.time()

    load = np.zeros(N)

    path = []

    for row in range(M):

        matrix[row] += load

        tmp\_row = matrix[row].copy()

        e, i = np.min(tmp\_row), np.argmin(tmp\_row)

        load[i] = e

        path.append(i)

    #return load, path, time.time()-start

    return load, np.max(load)

def main():

    M = int(input("M: "))

    N = int(input("N: "))

    min\_val = int(input("Min: "))

    max\_val = int(input("Max: "))

    num\_massives = 100

    win\_p2, win\_p3, win\_zver = 0, 0, 0

    all\_p2, all\_p3, all\_zver = [], [], []

    for i in range(num\_massives):

        # Генерация случайного массива

        matrix = np.array([[random.randint(min\_val, max\_val) for \_ in range(N)] for \_ in range(M)])

        random\_sorted\_matrix = matrix

        asc\_sorted\_matrix = sort(matrix, 1)

        desc\_sorted\_matrix = sort(matrix, 2)

        # Выполнение алгоритма для разных вариантов сортировки

        rand\_loads\_p2, max\_rand\_p2 = alg(random\_sorted\_matrix, N, M, 2)

        asc\_loads\_p2, max\_asc\_p2 = alg(asc\_sorted\_matrix, N, M, 2)

        desc\_loads\_p2, max\_desc\_p2 = alg(desc\_sorted\_matrix, N, M, 2)

        all\_p2.append(max\_desc\_p2)

        rand\_loads\_p3, max\_rand\_p3 = alg(random\_sorted\_matrix, N, M, 3)

        asc\_loads\_p3, max\_asc\_p3 = alg(asc\_sorted\_matrix, N, M, 3)

        desc\_loads\_p3, max\_desc\_p3 = alg(desc\_sorted\_matrix, N, M, 3)

        all\_p3.append(max\_desc\_p3)

        rand\_loads\_zver, max\_rand\_zver = Plotnikov\_Zverev(random\_sorted\_matrix.tolist(), N, M)

        asc\_loads\_zver, max\_asc\_zver = Plotnikov\_Zverev(asc\_sorted\_matrix.tolist(), N, M)

        desc\_loads\_zver, max\_desc\_zver =Plotnikov\_Zverev(desc\_sorted\_matrix.tolist(), N, M)

        all\_zver.append(max\_desc\_zver)

        # Определение победителя по максимальной нагрузке

        max\_val\_res = np.min([np.max(desc\_loads\_p2), np.max(desc\_loads\_p3), np.max(desc\_loads\_zver)])

        if max\_val\_res == np.max(desc\_loads\_p2):

            win\_p2 += 1

        elif max\_val\_res == np.max(desc\_loads\_p3):

            win\_p3 += 1

            print('3', desc\_sorted\_matrix, desc\_loads\_p3, desc\_loads\_p2, np.max(desc\_loads\_p3), np.max(desc\_loads\_p2))

        elif max\_val\_res == np.max(desc\_loads\_zver):

            win\_zver += 1

    # Вычисление среднего значения нагрузки

    for i in [all\_p2, all\_p3, all\_zver]:

        aver = sum(i)/len(i)

        i.append(aver)

    # Вывод результатов

    print("---------------Lab 4---------------")

    print('-------Квадратичный критерий-------')

    print(all\_p2[-1], '|', win\_p2)

    print('-------Кубический критерий---------')

    print(all\_p3[-1], '|', win\_p3)

    print('-------Минимаксный критерий---------')

    print(all\_zver[-1], '|', win\_zver)

main()

**Вычислительные эксперименты**

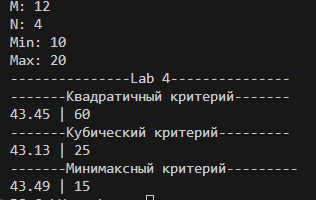


Рисунок 3 – Эксперимент №1

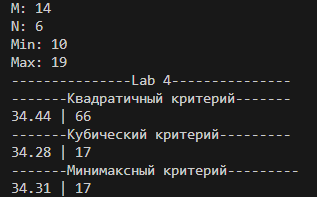


Рисунок 4 – Эксперимент №2

**Вывод**

Проанализировав данные, полученные входе экспериментов, можно сделать вывод, что модифицированный алгоритм квадратичного критерия является более эффективным, т.к. точность решения в данном случае несколько лучше, чем алгоритм кубического и минимаксного критерия.

**Литература**

1. Коффман Э.Г. “Теория расписания и вычислительные машины” – M.: “Наука”, 1987

2. Романовский И.В. “Алгоритмы решения экстремальных задач” – М.: “Наука”, 1977

3. Пашкеев С.Д., Минязов Р.И., Могилевский В.Д. “Машинные методы оптимизации в технике связи” – М.: “Связь”, 1976.