

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 «Эвристические методы и алгоритмы»

**Выполнила:**

студентка уч. гр. ВМО31

Басенко Белла Валерьевна

**Проверил:**

проф. Кобак В. Г.

Ростов-на-Дону

2024

**Введение**

Задачи проектирования и управления в системах, для которых необходимо распределение работы между параллельно работающими разнородными вычислительными устройствами занимают значимое место в теории построения расписаний. Практическая актуальность таких задач определяется существенными возможностями экономии машинного времени и вытекающими функциональными и эксплуатационными преимуществами.

Теоретическая сложность нахождения наилучшего распределения связана с необходимостью решения экстремальных задач комбинаторного типа, требующих больших вычислительных ресурсов, так что эффект от нахождения близкого к оптимальному, с точки зрения времени выполнения, распределения может быть сведен на нет затратами на его получение.

В настоящем руководстве приводятся методы получения расписаний, приводящие к небольшим затратам на вычисление за счет отказа от получения оптимального решения, но в тоже время позволяющие найти приемлемое решение, близкое к оптимальному.

**Постановка задачи**

Имеется  независимых работ , которые необходимо распределить на  параллельно работающих разнородных устройств  по критерию , где - время завершения работы процессора . Каждое устройство  выполняет только одну работу в определенный момент времени и выполнение задания не прерывается для передачи на другой процессор. Известно (вес) время выполнения  задания  на любом из устройств . Требуется найти такое распределение заданий по процессорам, при котором суммарное время выполнения заданий на каждом из процессоров было бы минимальным.

Получение оптимального распределения в такой постановке приводит к громоздким вычислениям, требующим значительного времени машинного счета, поэтому цель – продемонстрировать алгоритмы, с помощью которого можно находить с малыми затратами достаточно приемлемое решение.

**Алгоритмы распределения работ на параллельно работающие устройства**

Описанный ниже метод более эффективен по скорости поиска приемлемого по точности решения.

Модифицированный алгоритм построения расписания с произвольной загрузкой:

Дана прямоугольная матрица .

Ш.1 Упорядочим строки матрицы *T* по убыванию сумм всех их элементов.

Ш.2 В преобразованной матрице *T’* первой строке проведем расчёт согласно формуле (1):

Выберем минимальный элемент T. Индекс данного элемента примем за индекс элемент распределения и прибавим элемент строки с данным индексом к соответствующему элементу следующей строки.

Ш.3 Следующая строка теперь учитывает предыдущее решение. Выполним расчёт элементов второй строки по формуле (1), найдем индекс минимального элемента T, прибавим элемент второй строки с данным индексом к соответствующему элементу третьей строки и т.д.

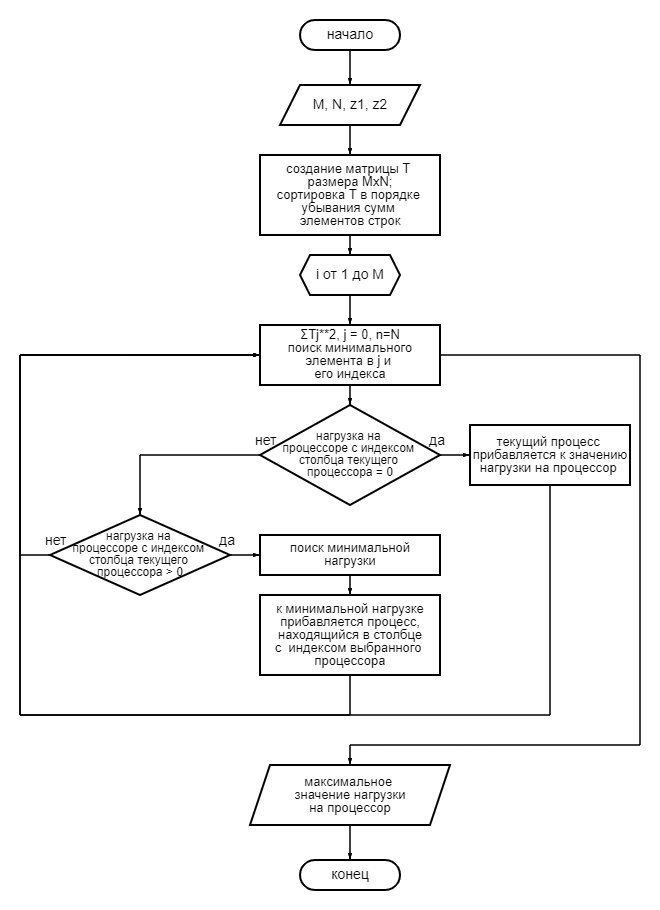
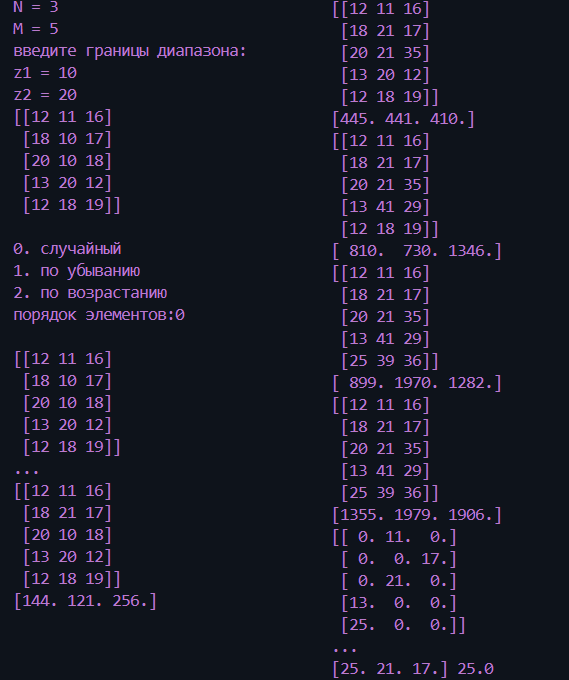


Рисунок 1 – Схема модифицированного алгоритма построения расписания с произвольной загрузкой

**Вычислительные эксперименты**

****

**Листинг программы**

import random, numpy as np

N, M = int(input("N = ")), int(input("M = "))

print("введите границы диапазона:")

z1, z2 = int(input("z1 = ")), int(input("z2 = "))

temp = np.array([[random.randint(z1, z2) for j in range(N)] for i in range(M)])

print(temp)

def sort(T):

    choice = int(input("\n0. случайный\n1. по убыванию\n2. по возрастанию\nпорядок элементов:"))

    match choice:

        case 0: return T

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

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

def alg(T, N, M):

    print(), print(T)

    input("...")

    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, 2)

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

                Tt = Tt + lead[i]\*\*2

                Tt[i] -= lead[i]\*\*2

            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, 2)

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

                Tt = Tt + lead[i]\*\*2

                Tt[i] -= lead[i]\*\*2

            ind = np.argmin(Tt)

            elem = current\_row[ind]

            mins[row][ind] = elem

            lead[ind] = elem

        print(matrix)

        print(Tt)

    print(mins), input("...")

    tasks = np.array([])

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

    print(tasks, max(tasks))

T = sort(temp)

alg(T, N, M)

**Вывод**

Проанализировав данные, полученные входе экспериментов, можно сделать вывод, что модификация алгоритма построения расписания с произвольной загрузкой более эффективна по скорости поиска подходящего по точности решения, чем классическая версия алгоритма.

Литература

1. Поспелов Д.А. “Введение в теорию вычислительных машин” – M.: “Советское радио”, 1972

2. Пашкеев С.Д. “Основы мультипрограммирования для специализированных вычислительных машин” – M.: “Советское радио”, 1964

3. Плотников В.Н., Зверев В.Ю. “Техническая кибернетика №3” M., 1974

4. Бондаренко А.Т., Сапатый П.С. “Техническая кибернетика №4” –Киев, 1975