Федеральное государственное бюджетное образовательное учреждение
высшего образования «Сибирский государственный университет
телекоммуникаций и информатики»

ОТЧЁТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №4

по дисциплине
«Теория функционирования распределённых вычислительных систем»

Выполнил:
Студент гр. ИВ-622
Тимофеев Д.А.

Проверила:
Преподаватель Кафедры ВС
Ткачева Т.А.

Новосибирск 2020

Оглавление

1.Цель роботы

1. Написать программу, реализующую алгоритмы NFDH и FFDH [1, 2] для приближенного решения задачи (1) – (5).

В качестве входных параметров программа получает имя файла с набором задач, количество n ЭМ в системе и название алгоритма (NFDH или FFDH). Результат работы программы: расписание S решения задач, значение T(S) целевой функции, отклонение ε значения целевой функции от её нижней границы T′, время t выполнения алгоритма в секундах.

$$\varepsilon = \frac{T(S) - T'}{T'}, \quad T' = \frac{1}{n} \sum_{j \in J} r_j t_j \ .$$

Упорядочивание задач в алгоритмах NFDH и FFDH выполнять сортировкой подсчетом (Counting Sort).

Для реализации алгоритма FFDH с вычислительной сложностью O(nlogn) рекомендуется использовать бинарное дерево турнира (tournament tree, max winner tree). Каждый лист дерева – это уровень упаковки, а значение листа – это количества свободных ЭМ на уровне. Каждый внутренний узел дерева содержит максимальное из значений левого и правого дочерних узлов. В таком дереве поиск первого подходящего уровня (First Fit) выполняется за время O(logn).

2. Исследовать время выполнения алгоритмов в зависимости от количества m задач в наборе.

Сформировать 10 наборов задач с m = 500, 1000, …, 5000; параметры задач генерировать как равномерно распределенные псевдослучайные числа rj ∈{1, 2, …, n}, tj ∈{1, 2, …, 100}. Рассмотреть случаи n = 1024, 4096.

Ответить на вопросы:

• Какова вычислительная сложность алгоритмов?

• Как зависит время работы алгоритмов от значения параметра m?

• Как зависит время работы алгоритмов от значения параметра n?

3. Провести сравнительный анализ значений целевой функции от расписаний, формируемых алгоритмами.

Сформировать 10 наборов задач (m = 500, 1000, …, 5000); параметры задач генерировать как равномерно распределенные псевдослучайные числа rj ∈{1, 2, …, n}, tj ∈{1, 2, …, 100}. Во всех 10 экспериментах n = 1024. По результатам экспериментов построить оценки математического ожидания и среднеквадратического отклонения случайной величины ε.

Ответить на вопрос: какой из алгоритмов, на рассмотренных наборах задач, формировал более точные расписания?

2.Теория

2.1Определения основных параметров и основные формулы

Имеется распределенная вычислительная система (ВС) укомплектованная n элементарными машинами (ЭМ). Задан набор из m параллельных задач. Каждая задача j ∈ J = {1, 2, …, m} характеризуется временем tj решения и количеством rj элементарных машин необходимых для неё.

Требуется построить расписание S решения параллельных задач на распределенной ВС. Для каждой задачи необходимо определить момент времени τj начала решения её ветвей и их распределение по элементарным машинам. Пусть xji ∈ C = {1, 2, …, n} – номер ЭМ, на которую распределена ветвь i ∈ {1, 2, …, rj} задачи j ∈ J.

Обозначим через J(t) = {j ∈ J | τj ≤ t ≤ τj + tj} множество задач, решаемых на распределенной ВС в момент времени t.

Будем называть расписание S = (τ1, τ2, …, τm; x11, x12, …, 1 1 r x , …, xm1, xm2, …, mrm x) допустимым, если оно удовлетворяет ограничениям.

1) В любой момент времени на ресурсах распределенной ВС решается не более n ветвей параллельных задач:

$$\sum_{j \in J(t)} r_j \le n, \ \forall t \in \boldsymbol{R}.$$

2) Ветви параллельных задач решаются на разных элементарных машинах:

$$\prod_{j \in J(t)} \prod_{j' \in J(t) \setminus \{j\}} (x_{ji} - x_{j'i'}) \neq 0, \ \forall t \in \boldsymbol{R}, \ i = 1, 2, \ldots, r_j, i' = 1, 2, \ldots, r_{j'}.$$

Обозначим через Ω множество допустимых расписаний. В качестве показателя оптимальности расписаний будем использовать время T(S) окончания решения последней задачи

$$T(S) = \max_{j \in J} \{\tau_j + t_j\}.$$

Итак, требуется найти допустимое расписание S ∈ Ω, доставляющее минимум целевой функции T(S). Формально

$$T(S) = \max_{j \in J} \{\tau_j + t_j\} \to \min_{S \in \Omega} \qquad (1)$$

При ограничениях:

$$\sum_{j \in J(t)} r_j \le n, \ \forall t \in \boldsymbol{R}, \qquad (2)$$

$$\prod_{j \in J(t)} \prod_{j' \in J(t) \setminus \{j\}} (x_{ji} - x_{j'i'}) \neq 0, \ \forall t \in \boldsymbol{R}, \ i = 1, 2, \ldots, r_j, i' = 1, 2, \ldots, r_{j'}, \qquad (3)$$

$$x_{ji} \in C, \ j = 1, 2, \ldots, m, \ i = 1, 2, \ldots, r_j, \qquad (4)$$

$$\tau_j \in \boldsymbol{R}, \ j = 1, 2, \ldots, m. \qquad (5)$$

Задача (1) − (5) относится к дискретной оптимизации и является трудноразрешимой.

Один из подходов к приближенному решению задачи (1) − (5) основан на её сведении к задаче двумерной упаковки прямоугольников в полуограниченную полосу (2D Strip Packing, 2DSP). Параллельная задача j ∈ J представляется в виде прямоугольника шириной rj и высотой tj условных единиц. Ширина полосы – n условных единиц, высота не ограничена. Требуется упаковать прямоугольники без их вращений и пересечений в полосу так, чтобы высота упаковки была минимальной. На рис. 1 приведен пример упаковки 5 прямоугольников (задач) в полосу шириной 10 условных единиц (элементарных машин).

Задача 1 запускается на решение в нулевой момент времени и использует элементарные машины 1 − 5, задача 4 начинает решаться в момент времени 2 и использует ЭМ 8, 9, 10. Значение целевой функции T(S) = 8.

Рис. 1. Пример упаковки прямоугольников в полуограниченную полосу $m = 5; n = 10; T(S) = 8$

3.Ход работы

Была написана программа, которая реализует алгоритмы NFDH и FFDH. Так же были проведены эксперименты для сравнительного анализа и времени выполнения некоторой задачи. Задача подразумевает собой генерировать равномерно распределенные псевдослучайные числа. При разных n – 1024, 4096.

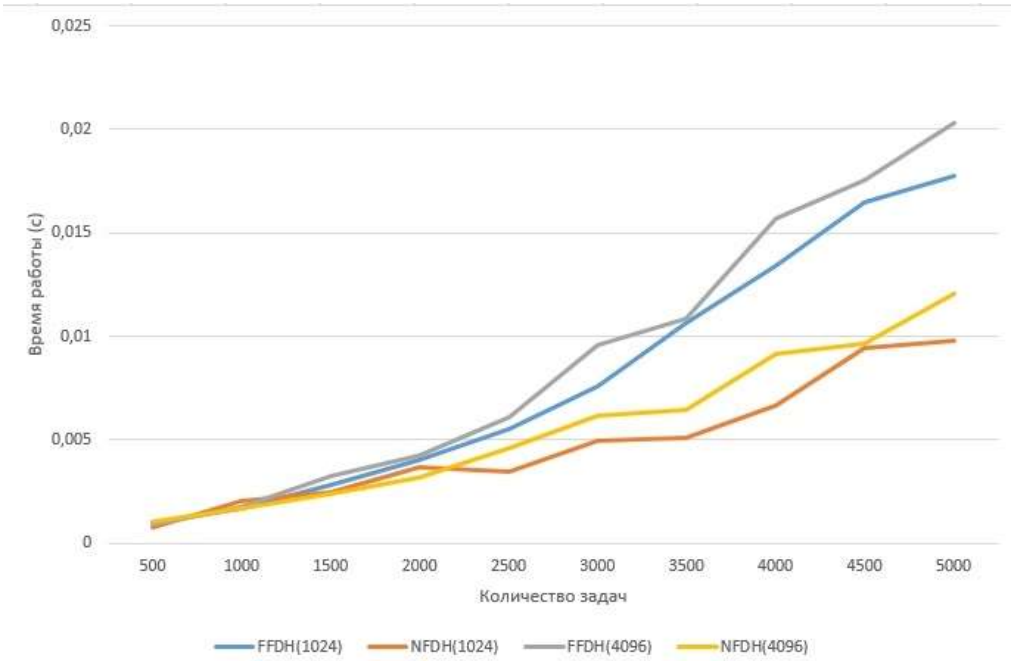

Рисунок 1График выполнения задач алгоритмами

4.Заключение

В результате лабораторной работы были реализованы алгоритмы NFDH и FFDH. Так же проведены экспериментальные нагрузки на каждый из этих алгоритмов для выявления сравнительной оценки (времени выполнения).

FFDH упаковывает элемент R (в не увеличивающейся высоте) на первом уровне , где R подходит. Если ни один уровень не может вместить R, создается новый уровень. Временная сложность FFDH O (n * log n). Аппроксимация FFDH (I) <= (17/10) · OPT (I) +1; асимптотическая граница 17/10 является плотной.

NFDH упаковывает следующий элемент R (в возрастающей высоте) на текущий уровень, если R подходит. В противном случае текущий уровень закрыт и создается новый уровень. Сложность времени O (n * log n). Отношения приближения NFDH (I) <= 2 · OPT (I) +1; асимптотическая оценка 2 плотна.

Из графика можно сделать вывод, что алгоритм FFDH эффективнее , чем NFDH. Алгоритм FFDH в равных условиях с NFDH выполнил задания быстрее , с увеличением количества входных данных время на выполнения задач растет пропорционально.

5. Листинг

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>

#include "../include/extralib.h"
#include "../include/color.h"

int main(int argc, char *argv[]) {
  if (argc < 4) {
    fprintf(stderr, "%s[ERROR]%s  Not enough arguments!%s\n", RED, YELLOW, RESET);
    exit(EXIT_FAILURE);
  }

  char* filename = argv[1];
  char* TDSP_method = argv[2];

  TT.cores = atoi(argv[3]);
  TT.time = 0;
  TT.occupancy = 0;
  TT.curr_core = 0;

  int tasks_amount = 0;

  int *list_of_tasks[2];
  list_of_tasks[0] = (int*)malloc(RANGE * sizeof(int));
  list_of_tasks[1] = (int*)malloc(RANGE * sizeof(int));

  FILE* fd_in = fopen(filename, "r");
  if (fd_in) {
    fscanf(fd_in, "%d", &tasks_amount);
    if (tasks_amount > RANGE) {
      fprintf(stderr, "%s[ERROR]%s  Amount of tasks is too large!%s\n", RED, YELLOW, RESET);
      fclose(fd_in);
      exit(EXIT_FAILURE);
    }

    for (int i = 0; i < tasks_amount; ++i) {
      fscanf(fd_in, "%d", &list_of_tasks[0][i]);
      fscanf(fd_in, "%d", &list_of_tasks[1][i]);
    }

    fclose(fd_in);
  }

  if (strcmp(TDSP_method, "NFDH") == 0) {
    printf("%s[SYSTEM]%s  For %s%d%s tasks from %s%s%s and %s%d%s EM has been applied %s%s%s
method...%s\n", YELLOW, WHITE, GREEN, tasks_amount, WHITE, GREEN, filename, WHITE, GREEN, TT.cores, WHITE,
GREEN, TDSP_method, WHITE, RESET);
    NFDH(list_of_tasks, tasks_amount);
  } else if (strcmp(TDSP_method, "FFDH") == 0) {
    printf("%s[SYSTEM]%s  For %s%d%s tasks from %s%s%s and %s%d%s EM has been applied %s%s%s
method...%s\n", YELLOW, WHITE, GREEN, tasks_amount, WHITE, GREEN, filename, WHITE, GREEN, TT.cores, WHITE,
GREEN, TDSP_method, WHITE, RESET);
    FFDH(list_of_tasks, tasks_amount);
  } else {
    fprintf(stderr, "%s[ERROR]%s  Unknown method for use!%s\n", RED, YELLOW, RESET);
    exit(EXIT_FAILURE);
  }

  return EXIT_SUCCESS;
}
#include <iostream>
#include <fstream>
using namespace std;

void tasksGenerate(int n) {
        int m = 500;
        for (int i = 0; i < 10; i++) {
                string str = "./result/task" + to_string(i) + "n" + to_string(n);
                ofstream fout(str);

                fout << n << " " << m << endl;

                for (int i = 0; i < m; i++)
```

```
                        fout << 1 + rand() % 100 << " " << 1 + rand() % n << endl;

                fout.close();
                m += 500;
        }
}

int main(int argc, char* argv[]) {
        int n = 0;
        cin >> n;
        srand(time(0));
        tasksGenerate(n);

        return 0;
}
#include <iostream>
#include <fstream>
#include <utility>
#include <map>
#include <cmath>
#include <cstdlib>
#include <ctime>

using namespace std;

int n = 0; /* Число ЭМ в системе */
int m = 0; /* Число задач в наборе для обработки */

typedef struct TreeNode {
        int gap_size;/* Вместимость уровня */
        int start_time; /* Время запуска уровня */
        int type;
        struct TreeNode *left;
        struct TreeNode *right;
        struct TreeNode *top;
} TreeNode;

typedef struct tasks {
        int number,                     /* Номер задачи */
        rank,                           /* Ранг задачи */
        time,                           /* Время выполнения задачи */
        start_time,             /* Время запуска задачи */
        position;                       /* Смещение размещения задачи на ЭМ */
        struct tasks *next; /* Поле для сортировки подсчётом */
} tasks;

int NFDH_sort(tasks *root) {

        int currentLevel = 0, nextLevel = 0, freeMachineNumber = 0;
        tasks *ptr = root;

        currentLevel = 0;
        freeMachineNumber = ptr->rank;
        ptr->start_time = currentLevel;
        nextLevel = ptr->time;

        ptr = ptr->next;
        while (ptr->next != NULL)
        {
                if (ptr->rank + freeMachineNumber > n)
                {
                        currentLevel = nextLevel;
                        nextLevel += ptr->time;
                        ptr->start_time = currentLevel;
                        freeMachineNumber = ptr->rank;
                } else {
                        ptr->start_time = currentLevel;
                        freeMachineNumber += ptr->rank;
                }
                ptr = ptr->next;
        }

        return nextLevel;
}

TreeNode* FFDH_Create_Node(TreeNode *top, int i, int depth) {
        TreeNode *ptr = (TreeNode *)malloc(sizeof(TreeNode));
```

```
        if (i != 0)
                ptr->top = top;

        ptr->gap_size = n;
        if (i < depth) {
                ptr->left = FFDH_Create_Node(ptr, i+1, depth);
                ptr->right = FFDH_Create_Node(ptr, i+1, depth);
                ptr->type = 0;
        }
        else {
                ptr->start_time = -1;
                ptr->type = 1;
        }
        return ptr;
}

tasks* createNode(int number, int rank, int time) {
        tasks *ptr = (tasks *)malloc(sizeof(tasks));
        ptr->number = number;
        ptr->rank = rank;
        ptr->time = time;
        ptr->next = NULL;

        return ptr;
}

TreeNode* FFDH_Find_Node(TreeNode *root, int gapsize, int depth) {
        TreeNode *ptr = root, *b_ptr;
        do {
                if (ptr->left->gap_size >= gapsize)
                        ptr = ptr->left;
                else
                        ptr = ptr->right;
        } while (--depth);

        ptr->gap_size -= gapsize;
        b_ptr = ptr;
        while (ptr != root) {
                if (ptr->top->left->gap_size >= ptr->top->right->gap_size)
                        ptr->top->gap_size = ptr->top->left->gap_size;
                else
                        ptr->top->gap_size = ptr->top->right->gap_size;

                ptr = ptr->top;
        }
        return b_ptr;
}

int FFDH_Compute(tasks *data)
{
        int depth, time = 0;
        TreeNode *root, *ptr;
        tasks *tptr = data;

        depth = ceil(log(m) / log(2));
        root = FFDH_Create_Node(NULL, 0, depth);

        while (tptr->next != NULL) {
                ptr = FFDH_Find_Node(root, tptr->rank, depth);
                if (ptr->start_time == -1) {
                        ptr->start_time = time;
                        time += tptr->time;
                }
                tptr->start_time = ptr->start_time;
                tptr->position = (n - ptr->gap_size - tptr->rank);

                tptr = tptr->next;
        }
        return time;
}



int main(int argc, char* argv[]) {
        int a = 0, b = 0;
        int TS = 0;
                double Tsh = 0.0, e = 0;
```

```
        string str;

        if (argc < 2) {
                cout << "Enter filename";
                return 0;
        }
                ifstream fin(argv[1]);

        fin >> n >> m;

        int mass[n+1], mass2[n+1];
        int myMap1[m], myMap2[m], mySortMap1[m], mySortMap2[m];

        for (int i = 0; i < m; i++) {
                fin >> a >> b;
                myMap1[i] = a;
                myMap2[i] = b;
                Tsh += a*b;
        }

        Tsh /= n;
        for (int i = 0; i < n+1; i++) {
                mass[i] = 0;
                mass2[i] = 0;
        }

        for (int i = 0; i < m; i++)
                mass[myMap1[i]]++;

        mass2[0] = mass[0];
        for (int i = 1; i < n+1; i++)
                mass2[i] = mass2[i-1] + mass[i];

        mass[0] = 0;
        for (int i = 1; i < n+1; i++)
                mass[i] = mass2[i-1];

        for (int i = 0; i < m; i++) {
                mySortMap1[mass[myMap1[i]]] = myMap1[i];
                mySortMap2[mass[myMap1[i]]] = myMap2[i];
                mass[myMap1[i]]++;
        }

        tasks *root = createNode(0, mySortMap2[m-1], mySortMap1[m-1]);
        tasks *end = root;
        for (int i = m-2, j = 1; i >= 0; i--, j++) {
                end->next = createNode(j, mySortMap2[i], mySortMap1[i]);
                end = end->next;
        }

        unsigned int startTime =  clock();
        TS = NFDH_sort(root);
        unsigned int endTime =  clock() - startTime;
        e = (TS - Tsh)/Tsh;

        cout << m << " " << TS << endl;
        startTime = clock();
        TS = FFDH_Compute(root);
        endTime = clock() - startTime;
        e = (TS - Tsh)/Tsh;
        cout << m << " " << TS << endl;
        fin.close();
        return 0;
}
```