Задание №8 в рамках вычислительного практикума.
Обработка матриц

Студент: Краснов Леонид

Группа: ИУ7-21Б

## Оглавление



Вариант 3

Сортировка строк матрицы по сумме элементов в строке с кешированием сумм и без. Дополнительно рассмотреть плоскость с оптимизациями Os, O0, O1, O2, O3.

## Подготовка к тестированию

Так как матрица может быть не квадратной, то через define можно задать кол-во элементов в строке. Количество строк в матрице задается при вызове исполняемого файла. Матрица заполняется автоматически целыми числами.

**Функция для замерение времени**

```
// Получение времени в микросекундах
unsigned long long microseconds_now(void)
{
    struct timeval val;
    if (gettimeofday(&val, NULL))
    {
        return (unsigned long long) - 1;
    }
    return val.tv_sec * 1000000ULL + val.tv_usec;
}
```

**Автоматическое заполнение матрицы элементами**

```
void input_elements(int arr[][M], int n)
{
    srand(time(NULL));
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < M; j++)
        {
            int element = rand() % 10;
            arr[i][j] = element;
        }
    }
}
```

Сортировка матрицы с кешированием сумм

Целевая функция

```
void sort_matrix(int matrix[][M], int n)
{
    int array_sum[n];
    // Заполнение массива сумм
    for (int i = 0; i < n; i++)
    {
        array_sum[i] = sum_arr(matrix[i], M);
    }
    sort_2(array_sum, matrix, n);
}
```

Функции, которые используются в целевой

```
int sum_arr(const int arr[], int n)
{
```

```
    int sum = 0;
    for (int i = 0; i < n; i++)
    {
        sum += arr[i];
    }
    return sum;
}


// Сортировка массивов X и Y по X
void sort_2(int arr1[], int arr2[][M], int n)
{
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n - 1; j++)
        {
            if (arr1[j] > arr1[j + 1])
            {
                int s = arr1[j];
                int line[M];
                arr1[j] = arr1[j + 1];
                arr1[j + 1] = s;

                memcpy(line, arr2[j], sizeof(line));
                memcpy(arr2[j], arr2[j + 1], sizeof(line));
                memcpy(arr2[j + 1], line, sizeof(line));
            }

        }
    }
}
```

Сортировка матрицы без кеширования сумм

Целевая функция

```
void sort_matrix(int matrix[][M], int n)
{
    int sum_1 = 0;
    int sum_2 = 0;
    for (int i = 0 ; i < n - 1; i++)
    {
        //  Перебор всех строчек матрицы
        for (int j = 0; j < n - i - 1; j++)
```

```
        {
            // Это выражение - сумма элементов массива
            sum_1 = sum_arr(matrix[j], M);
            sum_2 = sum_arr(matrix[j + 1], M);
            if (sum_1 > sum_2)
            {
                swap(matrix[j], matrix[j + 1]);
            }
        }
    }
}
```

Функции, которые используются в целевой

```
void swap(int *a, int *b)
{
    int c[M];
    memcpy(c, a, sizeof(c));
    memcpy(a, b, sizeof(c));
    memcpy(b, c, sizeof(c));
}


int sum_arr(const int arr[], int n)
{
    int sum = 0;
    for (int i = 0; i < n; i++)
    {
        sum += arr[i];
    }
    return sum;
}
```

Скрипты для сравнения производительности

1. **build_apps.sh**

```
#!/bin/bash


opts="O1 O2 O3 O0 Os"
programms="programm_1 programm_2"
for i in $programms; do
    for opt in $opts; do
        gcc -std=c99 -Wall -Werror -Wpedantic -Wextra \
        -"${opt}" \
        "$i".c -o ./apps/"${i}"_"${opt}".exe
```

```
        echo "${i}"_"${opt}".exe готово
    done
done
```

2. update_data.sh

```
#!/bin/bash

programms="programm_1 programm_2"
sizes=""
opts="O1 O2 O3 O0 Os"
count=10

i=100

while [ $i -le 2000 ]
do
    sizes="$sizes $i"
    ((i+=100))
done

if [ ! -z $1 ]; then
    count=$1
fi
if [ ! -z $2 ]; then
    sizes=$2
fi
if [ ! -z $3 ]; then
    programms=$3
fi
if [ ! -z $4 ]; then
    opts=$4
fi


for prog in $programms; do
    for co in $(seq "$count"); do
        for opt in $opts; do
            for i in $sizes; do
                echo -n -e "${prog}_""${opt}_""${i}\t $co/$count \r"
                ./apps/"${prog}"_"${opt}".exe
"${i}">>./data/"${prog}"_"${opt}"_"${i}".txt
            done
        done
```

```
    done
done
```

3. **make_preproc.sh**

```
import os
import shutil


#  Функция принимает имя файла и возвращает массив целых чисел из этого файла
def read(str):
    with open(str, "r") as f:
        mass = []
        for line in f:
            a = int(line)
            mass.append(a)
    return mass


def find_sizes(folder):
    sizes = []
    files = os.listdir(folder)
    for i in range(len(files)):
        file = files[i]
        file = file[14:]
        file = file.replace(".txt", "", 1)
        sizes.append(int(file))
    sizes = list(set(sizes))
    sizes.sort()
    for i in range(len(sizes)):
        sizes[i] = str(sizes[i])
    # print(sizes)
    return sizes


#  Функция принимает массив и возвращает среднеарифметическое
def average(arr):
    sum = 0
    for i in range(len(arr)):
        sum += arr[i]
    return sum/len(arr)
```

```
#  Функция принимает массив и возвращает его медианное значение
def find_median(arr):
    arr.sort()
    # print(arr)
    if len(arr) == 1:
        return arr[0]
    if len(arr) % 2 == 0:
        return arr[len(arr)//2]
    return (int(arr[int(len(arr)/2) - 1]) + int(arr[int(len(arr)/2)])) / 2


#  Находит нижний квартиль
def finde_lower_quartile(arr):
    a = len(arr)
    #  граница 25%
    a //= 4
    return arr[a]


#  Находит верхний квартиль
def find_upper_quartile(arr):
    a = len(arr)
    #  Граница 75%
    a //= 4
    a *= 3
    return arr[a]


#  Функция удаляет подготовленные данные с прошлых эксперементов
def del_old(folder):
    shutil.rmtree(folder)
    os.mkdir(folder)


#  Записывает полученные данные в файл
def save_prep_data(stri, avg, med, mini, maxi, up_quart, low_quart):
    folder = './prep_data/'
    file = folder + stri
    with open(file, "w") as f:
        f.write(str(avg) + '\n')
        f.write(str(med) + '\n')
        f.write(str(mini) + '\n')
        f.write(str(maxi) + '\n')
```

```
        f.write(str(low_quart) + '\n')
        f.write(str(up_quart))
        file = "\r" + file



folder = './data/'
folder_out = './prep_data'

#  Удаляет предыдущие данные
del_old(folder_out)

programm = ['programm_1', 'programm_2']
option = ['O0', 'O1', 'O2', 'O3', 'Os']
for i in range(len(option)):
    option[i] = "_" + option[i]

size = find_sizes(folder)
for i in range(len(size)):
    size[i] = "_" + size[i]

ex = '.txt'
n = 0
print("Подготовка данных")
for prog in range(len(programm)):
    for opt in range(len(option)):
        for siz in range(len(size)):
            stri = programm[prog] + option[opt] + size[siz] + ex
            arr = read(folder + stri)
            avg = average(arr)
            med = find_median(arr)
            low_quart = finde_lower_quartile(arr)
            up_quart = find_upper_quartile(arr)
            mini = arr[0]
            maxi = arr[len(arr) - 1]
            save_prep_data(stri, avg, med, mini, maxi, up_quart, low_quart)
            n += 1
            progress = str(n) + ' / ' + str(len(programm) * len(option) * len(size))
+ '\r'
            print(progress, end="")
print()
```

**make_postproc.sh**

```
import os
import shutil

import matplotlib.pyplot as plt


# Обычный кусочно-линейный график зависимости времени выполнения
в любых единицах измерения времени от числа
# элементов массива для всех 10 вариантов программы.

#  Получение информации из переданной строки о том, что содержится в
файле
def whats_in_file(file):
    sp_file = file.split("_")
    prog_num = sp_file[1]
    opt_lvl = sp_file[2]
    col_elems = sp_file[3]
    col_elems = col_elems.partition('.')[0]
    file = 'post_prep_data_' + prog_num + '_' + opt_lvl + '.txt'

    return prog_num, opt_lvl, col_elems, file


#  Плучение информации из файла
def params_from_file(folder_input, file):
    val = []
    with open(folder_input + file) as f:
        for line in f:
            val.append(float(line))
    return val


#  Функция подготавливает данные для дальнейшего построения
графика
def linal_graph_data(files_input, folder_input, folder_out):
    print("____Таблица данных для кусочного графика____")
    print("col_elems\t time")
    files_input.sort()
    for file in files_input:
        avg = params_from_file(folder_input, file)[0]
        prog_num, opt_lvl, col_elems, file = whats_in_file(file)
        with open(folder_out + file, 'a') as f:
```

```
            f.write(str(col_elems) + ' ' + str(avg) + '\n')

    files_table = os.listdir(folder_out)
    files_table.sort()
    for file_tab in files_table:
        print("__tab data", file_tab, "__")
        with open(folder_out + file_tab) as f:
            for line in f:
                print(line, end="")


#  Сортировка массивов X и Y по X
def sort_2(arr1, arr2):
    for i in range(len(arr1)):
        for j in range(len(arr1) - 1):
            if arr1[j] > arr1[j + 1]:
                s = arr1[j]
                arr1[j] = arr1[j + 1]
                arr1[j + 1] = s
                s = arr2[j]
                arr2[j] = arr2[j + 1]
                arr2[j + 1] = s


#  Запись подготовленных данных в массивы по X и по Y для линейного графика
def linal_graph_to_mass(folder):
    x_array = []
    y_array = []
    files_input = os.listdir(folder)
    files_input.sort()
    for file in files_input:
        file = folder + file
        x_arg = []
        y_arg = []
        with open(file, "r") as f:
            for line in f:
                args = line.split(' ')
                x_arg.append(int(args[0]))
                y_arg.append(float(args[1]))
        sort_2(x_arg, y_arg)
        x_array.append(x_arg)
        y_array.append(y_arg)
```

```
    return x_array, y_array, files_input


#  Строит линейный график
def plot_linal_graph(x_array, y_array, labels_array):
    #  Удаление лишнего из лэйблов
    for i in range(len(labels_array)):
        labels_array[i] = labels_array[i].replace(".txt", "")
        labels_array[i] = labels_array[i].replace("post_prep_", "", 1)
        labels_array[i] = labels_array[i].replace("data", "prog")
    fig, ax = plt.subplots()
    #  Набор параметров для отображения графиков
    linestyle_array = ['-', '--', '-.', ':', (0, (3, 1, 1, 1, 1, 1))]
    markers = ['^', 's', 'X']
    #  Генерация 15 видов линий
    difference = []
    for j in range(len(markers)):
        for i in range(len(linestyle_array)):
            typ = [linestyle_array[i], markers[j]]
            difference.append(typ)
    #  Построение графиков
    for i in range(len(x_array)):
        ax.plot(x_array[i], y_array[i], label=labels_array[i],
linestyle=difference[i][0], marker=difference[i][1])

    #  Добавляем подписи к осям:
    ax.set_xlabel('Кол-во элементов')
    ax.set_ylabel('Время (микросекунды)')

    #  Добавление легенды, масштабной сетки и вывод графиков на поле
    ax.legend()
    ax.grid()

        #  Сохранение графика
    fig.savefig('line_plot.svg')


#  Функция удаляет подготовленные данные с прошлых эксперементов
def del_old(folder):
    shutil.rmtree(folder)
    os.mkdir(folder)
```

```
# Заданные папки
folder_in = "prep_data/"
folder_lingraph = "post_prep_lingraph_data/"
folder_data = 'data/'
# Массив в котором хранятся имена файлов из папки folder_in
files = os.listdir(folder_in)

# Линейный график

# Удаление старых данных
del_old(folder_lingraph)

# Подготовка данных для построения графика
linal_graph_data(files, folder_in, folder_lingraph)

# Построение графика по подготовленным данным
x, y, labels = linal_graph_to_mass(folder_lingraph)
plot_linal_graph(x, y, labels)
plt.show()
```

Кусочно-линейные графики зависимости времени выполнения от числа строк матрицы для всех вариантов программы

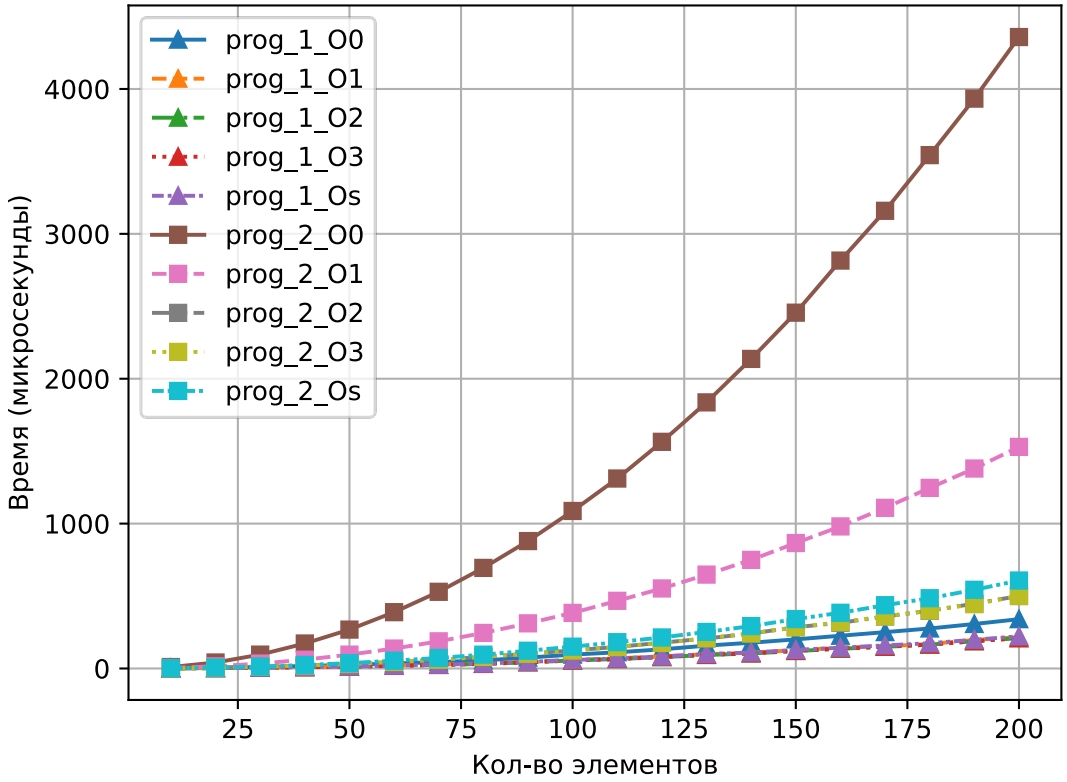


**Таблица к графику**

| plot_1_O0 | | plot_1_O1 | | plot_1_O2 | | plot_1_O3 | | plot_1_Os | |
|---|---|---|---|---|---|---|---|---|---|
| 10 | 2.3 | 10 | 1.2 | 10 | 0.7 | 10 | 0.7 | 10 | 0.6 |
| 20 | 5.5 | 20 | 3.1 | 20 | 2.5 | 20 | 2.4 | 20 | 2.7 |
| 30 | 10.5 | 30 | 6.5 | 30 | 5.6 | 30 | 5.3 | 30 | 5.6 |
| 40 | 17.3 | 40 | 10.3 | 40 | 8.9 | 40 | 8.9 | 40 | 9.8 |
| 50 | 24.6 | 50 | 14.6 | 50 | 12.7 | 50 | 12.8 | 50 | 13.4 |
| 60 | 34.2 | 60 | 20.7 | 60 | 18.7 | 60 | 18 | 60 | 19.7 |
| 70 | 44.2 | 70 | 27.5 | 70 | 24.4 | 70 | 24.4 | 70 | 26 |
| 80 | 56 | 80 | 35 | 80 | 31.6 | 80 | 31.6 | 80 | 33.8 |
| 90 | 75.2 | 90 | 46.6 | 90 | 42.8 | 90 | 42.6 | 90 | 45.4 |
| 100 | 95.7 | 100 | 58.4 | 100 | 55.4 | 100 | 54.9 | 100 | 58.6 |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 110 | 111.8 | 110 | 70.4 | 110 | 65.3 | 110 | 65.4 | 110 | 68.8 |
| 120 | 132.2 | 120 | 84.1 | 120 | 78.1 | 120 | 79.4 | 120 | 82.8 |
| 130 | 157.7 | 130 | 98.2 | 130 | 93 | 130 | 94 | 130 | 99.2 |
| 140 | 178 | 140 | 109.1 | 140 | 103.9 | 140 | 104.3 | 140 | 110.5 |
| 150 | 200.9 | 150 | 126.6 | 150 | 120.2 | 150 | 121.7 | 150 | 127.6 |
| 160 | 226 | 160 | 142.9 | 160 | 136.3 | 160 | 138.1 | 160 | 143.4 |
| 170 | 250.9 | 170 | 153.8 | 170 | 149.6 | 170 | 149.7 | 170 | 159.6 |
| 180 | 276.2 | 180 | 169.1 | 180 | 164.8 | 180 | 165 | 180 | 174.5 |
| 190 | 307.1 | 190 | 196.8 | 190 | 187.8 | 190 | 188.7 | 190 | 198.5 |
| 200 | 340.4 | 200 | 216.6 | 200 | 209.6 | 200 | 208.7 | 200 | 222 |
| plot_2_O0 | | plot_2_O1 | | plot_2_O2 | | plot_2_O3 | | plot_2_Os | |
| 10 | 10 | 10 | 3.9 | 10 | 1 | 10 | 1.1 | 10 | 1.8 |
| 20 | 41.4 | 20 | 15.2 | 20 | 4.8 | 20 | 4.7 | 20 | 6.1 |
| 30 | 95.8 | 30 | 33.9 | 30 | 11.2 | 30 | 11.5 | 30 | 12.9 |
| 40 | 172.5 | 40 | 60.6 | 40 | 20.2 | 40 | 20 | 40 | 23.8 |
| 50 | 268.5 | 50 | 95.7 | 50 | 31.1 | 50 | 31.4 | 50 | 37 |
| 60 | 388.7 | 60 | 137.4 | 60 | 44.9 | 60 | 44.9 | 60 | 54 |
| 70 | 529.3 | 70 | 187.5 | 70 | 60.7 | 70 | 60.5 | 70 | 72.7 |
| 80 | 694.3 | 80 | 245.4 | 80 | 79.5 | 80 | 79.6 | 80 | 95.5 |
| 90 | 879.5 | 90 | 311.7 | 90 | 102.5 | 90 | 101.6 | 90 | 122.5 |
| 100 | 1087.2 | 100 | 383.6 | 100 | 124.7 | 100 | 124.2 | 100 | 150.5 |
| 110 | 1311.2 | 110 | 467.1 | 110 | 149.1 | 110 | 148 | 110 | 181.6 |
| 120 | 1564.5 | 120 | 552.4 | 120 | 178.2 | 120 | 176.3 | 120 | 214.2 |
| 130 | 1836.8 | 130 | 648 | 130 | 207.1 | 130 | 208.8 | 130 | 252.3 |
| 140 | 2136.7 | 140 | 749.7 | 140 | 241.8 | 140 | 241.6 | 140 | 292.5 |
| 150 | 2455.9 | 150 | 865.9 | 150 | 284.8 | 150 | 284.7 | 150 | 341.1 |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 160 | 2815.5 | 160 | 980.7 | 160 | 315.6 | 160 | 319.7 | 160 | 384.4 |
| 170 | 3159.1 | 170 | 1108.7 | 170 | 358.9 | 170 | 359.5 | 170 | 436.9 |
| 180 | 3543.4 | 180 | 1246.5 | 180 | 398.4 | 180 | 400.3 | 180 | 485 |
| 190 | 3933.3 | 190 | 1380.2 | 190 | 447.5 | 190 | 443.9 | 190 | 543.2 |
| 200 | 4358.5 | 200 | 1529.6 | 200 | 500.3 | 200 | 500.6 | 200 | 607.2 |

Таблица для результатов с уровнем оптимизации O2 по заданию

| n | t1 | t2 | Величина 1 | Величина 2 |
|---|---|---|---|---|
| 10 | 1 | 0.7 | 2.26303441 | 1.83650127 |
| 20 | 4.8 | 2.5 | 2.08969365 | 1.98901422 |
| 30 | 11.2 | 5.6 | 2.05007153 | 1.61040511 |
| 40 | 20.2 | 8.9 | 1.93384578 | 1.59337213 |
| 50 | 31.1 | 12.7 | 2.01418846 | 2.12219299 |
| 60 | 44.9 | 18.7 | 1.95591679 | 1.72597104 |
| 70 | 60.7 | 24.4 | 2.02059844 | 1.93642845 |
| 80 | 79.5 | 31.6 | 2.15740557 | 2.57576127 |
| 90 | 102.5 | 42.8 | 1.86073552 | 2.44912897 |
| 100 | 124.7 | 55.4 | 1.87499771 | 1.725025 |
| 110 | 149.1 | 65.3 | 2.04903427 | 2.05717949 |
| 120 | 178.2 | 78.1 | 1.87768827 | 2.18145339 |
| 130 | 207.1 | 93 | 2.09031721 | 1.49551258 |
| 140 | 241.8 | 103.9 | 2.37236453 | 2.11222001 |
| 150 | 284.8 | 120.2 | 1.5911181 | 1.94769441 |
| 160 | 315.6 | 136.3 | 2.12072596 | 1.53579064 |
| 170 | 358.9 | 149.6 | 1.82672542 | 1.69297126 |
| 180 | 398.4 | 164.8 | 2.14954661 | 2.41634296 |
| 190 | 447.5 | 187.8 | 2.17438521 | 2.14108661 |

| | | | | |
|---|---|---|---|---|
| 200 | 500.3 | 209.6 | | |