МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ (НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСТИТЕТ)

ЛАБОРАТОРНАЯ РАБОТА №3

по курсу операционные системы I семестр, 2021/22 уч. год

Студент *Сикорский Александр Александрович, группа М8О-208Б-20*
Преподаватель *Миронов Евгений Сергеевич*
Вариант *Многопоточное вычисление определителя матрицы*
Оценка
Дата
Подпись

Содержание

Репозиторий

`https://github.com/sikorskii/os/tree/master/lab3`

Постановка задачи

Задание: Вариант 21:
Многопоточное вычисление определителя матрицы.

Общие сведения о программе

Исходный код лежит в 5 файлах:

1. src/main.c
2. src/mtxutilc.c - выделение памяти под матрицы, взаимодействие с пользователем, генерация матриц, получение миноров, вычисление определителя.
3. src/tutils.c - реализация функций потоков. Создание, структура с аргументами, собственно задача потока.
4. headers/mtxutils.h - заголовочный файл для функций матриц.
5. headers/tutils.h - заголовочный файл для функций потоков.

Программа собирается с помощью CMakeList. Нужно получить один исполняемый файл, но при этом нужно прилинковать библиотеку pthread.

Общий метод и алгоритм решения

Программа получает аргументом из командной строки количество потоков которые должны будут считать определитель. Пользователь же указывает размерность матрицы. Можно заполнить матрицу вручную, а можно сгенерировать какую-нибудь заданной размерности. После этого программа не самым оптимальный образом считает размер задачи для каждого потока. А именно с какого по какой элемент строки он будет раскладывать матрицу, чтобы считать определитель.

Выделяется память под идентификаторы потоков и под структуры с аргументами для каждого. Структуры заполняются так, что каждый поток может обращаться к матрице по указателю, знает левую и правую границу области своей работы, в эту же структуру он после запишет свой ответ. Создание всех потоков с запуском происходят в отдельной функции. Функция потока и аргументы передаются в него как void*, внутри потока уже происходит приведение структуры к нормальному виду.

Само вычисление определителя рекурсивное. Этот алгоритм несложно распараллелить. Я видел и другие алгоритмы, но в некоторых из них нужно знать значение элемента, посчитанное раннее (в том числе другим потоком), поэтому я не уверен, что такой способ можно разделить между потоками, ведь хорошо, чтобы они работали параллельно и независимо друг от друга. Поэтому рекурсивно считаем миноры, раскладывая первую строку матрицы.

После того, как все потоки запущены, нужно дождаться завершения работы каждого и собрать финальый ответ из результатов каждого потока. После этого нужно очистить везде память.

В коде есть импровизированный таймер для бенчмарка. Работает он не всегда корректно и я не уверен, правильные ли численные он значения он выводит. Тем не менее он позволяет увидеть, что при линейном росте количества потоков время работы сокращается тоже примерно линейно.

Исходный код

mtxutils.h

```
//
// Created by aldes on 02.10.2021.
//

#ifndef LAB3_MTXUTILS_H
#define LAB3_MTXUTILS_H

#include <stdbool.h>
#include <stdio.h>
#include <time.h>
#include <stdlib.h>
#include <unistd.h>

typedef struct matrix {
    int **matrix;
    int size;
} mtx;

int getRandInt();

int** fillFromInput();

int** fillWithRand();
```

```
bool prompt();

mtx getMatrix();

mtx getEmptyMatrix(int n);

mtx getReducedMatrix(mtx *matrix, int i, int j);

void printMatrix(mtx* matrix);

void cleanMatrix(mtx* matrix);

long long calculateMinor(int i, int j, int a, mtx* matrix);

long long calculateDet(mtx* matrix, int leftBound, int rightBound);

#endif //LAB3_MTXUTILS_H
```

tutils.h

```
//
// Created by aldes on 08.10.2021.
//

#ifndef LAB3_TUTILS_H
#define LAB3_TUTILS_H
#include "mtxutils.h"

void *func(void *args);

typedef struct arguments {
    mtx *matrix;
    int left_bound;
    int right_bound;
    long long result;
} arg_t;


void createThreads(int threads_num, pthread_t *threads,
                   void *func, arg_t * targs);

void joinThreads(int threads_num, pthread_t *threads);
```

```
#endif //LAB3_TUTILS_H
```

main.c

```
#include <stdio.h>
#include <pthread.h>
#include <stdlib.h>
#include <sys/time.h>

#include "../headers/mtxutils.h"
#include "../headers/tutils.h"

signed main(signed argc, char** argv) {

    if (argc != 2) {
        printf("Not enough args. Please specify the number of threads: "
               "./executable <threads number>\n");
        exit(1);
    }

    int threads_num = (int)strtol(argv[1], NULL, 10);

    mtx matrix;
    matrix = getMatrix();
    printMatrix(&matrix);

    int columnsPerThread = matrix.size / threads_num;

    if (columnsPerThread < 1) {

        printf("There is too many threads for this matrix, "
               "number of threads reduced: %d -> %d\n", threads_num,
               matrix.size);
        threads_num = matrix.size;

    }

    columnsPerThread = matrix.size / threads_num;

    printf("Colums per thread: %d\n Threads: %d, Matrix.size: %d\n",
           columnsPerThread, threads_num, matrix.size);

    pthread_t *threads = calloc(threads_num, sizeof(pthread_t));
```

```
arg_t *thread_args = calloc(threads_num, sizeof(arg_t));


for (int i = 0; i < threads_num; i++) {
    thread_args[i].matrix = &matrix;
    thread_args[i].result = 0;
    thread_args[i].left_bound = i * columnsPerThread;
    thread_args[i].right_bound = thread_args[i].left_bound
            + columnsPerThread;
}

thread_args[threads_num - 1].left_bound = (threads_num - 1)
        * columnsPerThread;
thread_args[threads_num - 1].right_bound = matrix.size;

struct timeval stop, start;
gettimeofday(&start, NULL);

createThreads(threads_num, threads, &func, thread_args);

joinThreads(threads_num, threads);

long long ans = 0;

for (int i = 0; i < threads_num; i++) {
    ans += thread_args[i].result;
    printf("ans %d = %lld\n",i, thread_args[i].result);
}

gettimeofday(&stop, NULL);

printf("took %lu mcs\n", (stop.tv_sec - start.tv_sec) *
    100000 + stop.tv_usec - start.tv_usec);

free(thread_args);

printf("Multithreading result is %lld\n", ans);

cleanMatrix(&matrix);

free(threads);
```

```
}
```

mtxutils.c

```
//
// Created by aldes on 02.10.2021.
//
#include "../headers/mtxutils.h"

#define MAX_ELEMENT 10

int getRandInt() {
    return rand() % MAX_ELEMENT;
}

int** fillWithRand(int n) {
    printf("Matrix will be generated with random\n");
    int **matrix = malloc(n * sizeof(int*));

    for (int i = 0; i < n; i++) {

        matrix[i] = malloc(n * sizeof(int));

        for (int j = 0; j < n; j++)

            matrix[i][j] = getRandInt();

    }

    return matrix;
}

int** fillFromInput(int n) {
    int **matrix = malloc(n * sizeof(int*));
    printf("Enter %d elements of matix:\n", n * n);
    for (int i = 0; i < n; i++) {

        matrix[i] = malloc(n * sizeof(int));

        for (int j = 0; j < n; j++) {

            scanf("%d", &matrix[i][j]);
```

```
        }
    }

    return matrix;
}

bool prompt() {
    printf("Fill mtx with random values?\n Enter 1/0 \n");
    int mode;
    scanf("%d", &mode);

    switch (mode) {
        case 1 :
            return true;

        case 0 :
            return false;

        default :
            return prompt();
    }
}

mtx getMatrix() {
    int n;
    printf("Enter mtx's dimension: \n");
    scanf("%d", &n);
    mtx matrix;
    matrix.size = n;
    matrix.matrix = NULL;

    if (prompt()) {

        srand(time(NULL));
        matrix.matrix = fillWithRand(n);

    }
    else {

        matrix.matrix = fillFromInput(n);

    }
```

```
    return matrix;

}

mtx getEmptyMatrix(int n) {
    mtx matrix;
    matrix.size = n;
    matrix.matrix = malloc(n * sizeof(int*));

    for (int i = 0; i < n; i++)
        matrix.matrix[i] = malloc(n * sizeof(int));


    return matrix;

}

mtx getReducedMatrix(mtx *matrix, int i, int j) {

    mtx newMatrix = getEmptyMatrix(matrix->size - 1);

    for (int a = 0, x = 0; a < matrix->size - 1; a++, x++) {

        for (int b = 0, y = 0; b < matrix->size - 1; b++, y++) {

            if (x == i)
                x++;
            if (y == j)
                y++;

            newMatrix.matrix[a][b] = matrix->matrix[x][y];
        }
    }

    return newMatrix;

}

void printMatrix(mtx *matrix) {
    printf("-----------------Matrix %d by %d-----------------\n",
            matrix->size, matrix->size);
```

```
    for (int i = 0; i < matrix->size; i++) {

        for (int j = 0; j < matrix->size; j++){

            printf("%5d ", matrix->matrix[i][j]);

        }

        printf("\n");

    }
}

void cleanMatrix(mtx* matrix) {
    for (int i = 0; i < matrix->size; i++)
        free(matrix->matrix[i]);

    free(matrix->matrix);
}

long basicDeterminant(mtx *matrix) {

    long long d = 0;

    if (matrix->size == 2) {
        return matrix->matrix[0][0] * matrix->matrix[1][1]
            - matrix->matrix[0][1] * matrix->matrix[1][0];
    }

    for (int i = 0; i < matrix->size; i++)
        d += calculateMinor(i, 0, matrix->matrix[i][0], matrix);

    return d;
}

long long calculateMinor(int i, int j, int a, mtx* matrix) {
    int degree;
    mtx submatrix = getReducedMatrix(matrix, i, 0);

    if ((i + j) % 2 == 0)
        degree = 1;
```

```
    else
        degree = -1;

    long det = degree * a * basicDeterminant(&submatrix);

    cleanMatrix(&submatrix);

    return det;
}

long long calculateDet(mtx* matrix, int leftBound, int rightBound) {
    long long det = 0;

    if(matrix->size == 1) {
        return matrix->matrix[0][0];
    }

    else if(matrix->size == 2) {
        return matrix->matrix[0][0] * matrix->matrix[1][1]
                - matrix->matrix[0][1] * matrix->matrix[1][0];
    }

    else {

        for(int j = leftBound; j < rightBound; j++) {

            long long temp = calculateMinor(j, 0, matrix->matrix[j][0], matrix);
            det += temp;
        }
    }

    return det;
}
```

tutils.c

```
//
// Created by aldes on 08.10.2021.
//

#include <pthread.h>
#include "../headers/tutils.h"
```

```
void *func(void *args) {
    arg_t *args_s = (arg_t*) args;
    printf("This is thread %lu\n", pthread_self());
    printf("My matrix size is %d, my right bound is %d\n",
           args_s->matrix->size, args_s->right_bound);

    args_s->result = calculateDet(args_s->matrix,
                                  args_s->left_bound,args_s->right_bound);
    pthread_exit(NULL);
}

void createThreads(int threads_num, pthread_t *threads, void *func, arg_t *targs) {
    for (int i = 0; i < threads_num; i++) {

        if (pthread_create(&threads[i], NULL, func, (void*)&targs[i]) != 0) {

            printf("Unable to create %d-th thread\n", i);
            free(threads);
            free(targs);
            exit(1);

        }

    }
}

void joinThreads(int threads_num, pthread_t *threads) {
    for (int i = 0; i < threads_num; i++) {

        if (pthread_join(threads[i], NULL) != 0) {

            printf("Unable to join %d-th thread\n", i);
            free(threads);
            exit(1);

        }
    }
}
```

Демонстрация работы программы

Ниже приведен пример работы программы.

>"string" обозначает пользовательский ввод.

test.txt

```
>10
>1
```

Вывод для десяти потоков и матрицы 10x10

```
Enter mtx dimension:
Fill mtx with random values?
Enter 1/0
Matrix will be generated with random
-----------------Matrix 10 by 10-----------------
7 8 4 8 3 5 4 5 6 4
0 6 3 1 6 2 9 9 6 5
9 2 3 1 8 5 4 0 2 9
3 1 0 9 2 5 7 8 0 3
5 1 1 8 2 8 0 2 9 8
9 1 2 2 4 3 0 9 3 4
0 8 6 0 8 0 6 5 8 6
8 3 7 1 2 2 1 4 6 1
3 7 4 5 2 8 8 2 9 3
6 0 4 4 0 2 4 6 9 5
Colums per thread: 1
Threads: 10, Matrix.size: 10
This is thread 140459259749952
My matrix size is 10, my right bound is 1
This is thread 140459251357248
My matrix size is 10, my right bound is 2
This is thread 140459242964544
My matrix size is 10, my right bound is 3
This is thread 140459217786432
My matrix size is 10, my right bound is 6
This is thread 140459201001024
My matrix size is 10, my right bound is 8
This is thread 140458986567232
My matrix size is 10, my right bound is 9
This is thread 140459226179136
My matrix size is 10, my right bound is 5
This is thread 140459234571840
```

```
My matrix size is 10, my right bound is 4
This is thread 140458978174528
My matrix size is 10, my right bound is 10
This is thread 140459209393728
My matrix size is 10, my right bound is 7
ans 0 = -24506510
ans 1 = 0
ans 2 = 4991112
ans 3 = 8659791
ans 4 = -104486045
ans 5 = 257778621
ans 6 = 0
ans 7 = -220619656
ans 8 = 69795066
ans 9 = 94653732
took 112655 mcs
Multithreading result is 86266111
```

Выводы

В ходе работы я познакомился с многопоточными вычислениями. Главной сложностью лабораторной было не создание потоков или работа с ними, а именно написание функций для работы с матрицами.