МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

Институт №8 «Компьютерные науки и прикладная математика»
Кафедра 806 «Вычислительная математика и программирование»

Лабораторная работа №2
по курсу «Операционные системы»

Выполнил: А. А. Устинов
Группа: М8О-207БВ-24
Преподаватель: Е. С. Миронов

Москва, 2025

Условие

Цель работы: Приобретение практических навыков в управлении потоками в ОС и обеспечении синхронизации между потоками

Задание: Составить программу на языке Си, обрабатывающую данные в многопоточном режиме. При обработки использовать стандартные средства создания потоков операционной системы (Windows/Unix). Ограничение максимального количества потоков, работающих в один момент времени, должно быть задано ключом запуска вашей программы. Так же необходимо уметь продемонстрировать количество потоков, используемое вашей программой с помощью стандартных средств операционной системы. В отчете привести исследование зависимости ускорения и эффективности алгоритма от входных данных и количества потоков. Получившиеся результаты необходимо объяснить.

Вариант: 12

Метод решения

Задача заключается в многопоточной обработке матрицы вещественных чисел с применением фильтров эрозии и наращивания. Программа распределяет строки матрицы между потоками и измеряет время выполнения для различных чисел потоков.

Архитектура программы

```
Lab2/
├── inc/
│   ├── filter_worker.h
│   ├── timer.h
│   ├── matrix_filters.h
│   └── matrix_utils.h
├── src/
│   ├── filter_worker.c
│   ├── main.c
│   ├── matrix_filters.c
│   └── matrix_utils.c
└── Makefile
```

Описание программы

Файл `main.c`

Назначение: Главный модуль программы, отвечающий за инициализацию и запуск основного алгоритма.

- Считывает аргументы командной строки: размеры матрицы M и N, количество повторов фильтров K, а также максимальное число одновременно работающих потоков T_{max}.
- Передаёт эти параметры функции `run_matrix_filters()`, определённой в модуле `matrix_filters.c`.

- Возвращает код завершения программы.

—

Файл `matrix_filters.c`

Основная логика программы:

- Реализует функцию `run_matrix_filters()`, выполняющую весь процесс вычислений.
- Инициализирует исходную матрицу случайными значениями.
- Запускает цикл по количеству потоков: $1, 2, 4, \ldots, T_{max}$.
  - Делит матрицу по строкам между потоками.
  - Создаёт заданное количество потоков.
  - В каждом потоке вызывает `apply_filter_static()` для обработки выделенного диапазона строк.
  - После завершения всех потоков измеряет время выполнения.
  - Записывает результаты в файл `results.csv` для последующего построения графика зависимости времени от числа потоков.

—

Файл `filter_worker.c`

Функции и логика:

- `apply_filter_static()` — функция, выполняемая каждым потоком. Применяет фильтры эрозии и наращивания K раз для своего диапазона строк.
- `min_in_3x3()` и `max_in_3x3()` — вычисляют минимальные и максимальные значения в окне 3×3 вокруг каждого элемента матрицы.

Структуры данных:

- `ThreadWork` — структура, содержащая указатели на исходную и результирующую матрицы, количество итераций фильтрации и диапазон строк, обрабатываемых данным потоком.

—

Файл `matrix_utils.c`

Функции:

- `create_matrix()` — выделяет память под матрицу заданного размера.
- `fill_matrix()` — заполняет матрицу случайными вещественными числами.
- `free_matrix()` — освобождает выделенную память.

Результаты

Разработанное решение представляет собой программу на языке C, выполняющую многопоточную обработку матрицы вещественных чисел с применением фильтров эрозии и наращивания. Программа позволяет измерять время выполнения для различного числа потоков и строить график зависимости времени от количества потоков. В результаты был получен графики зависимости времени от количества потоков и ускорения от количества потоков:

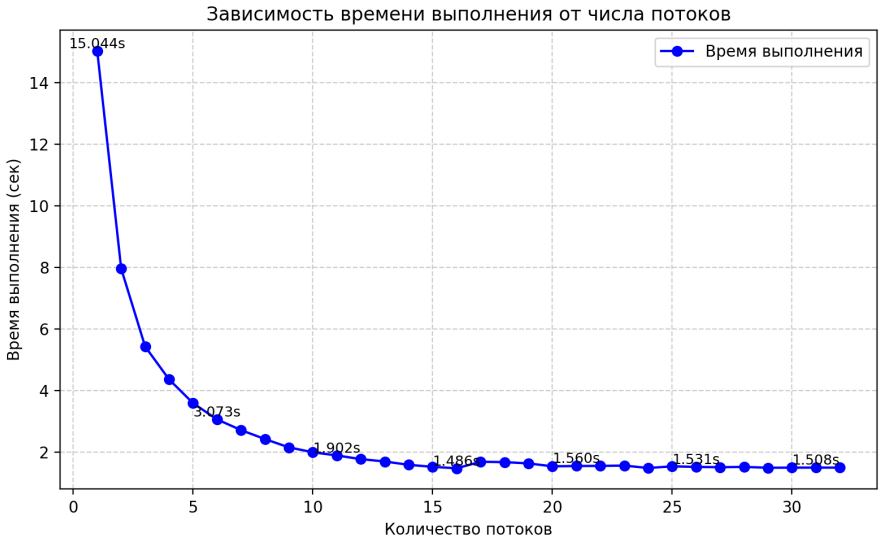

Рис. 1: График зависимости времени выполнения от количества используемых потоков.

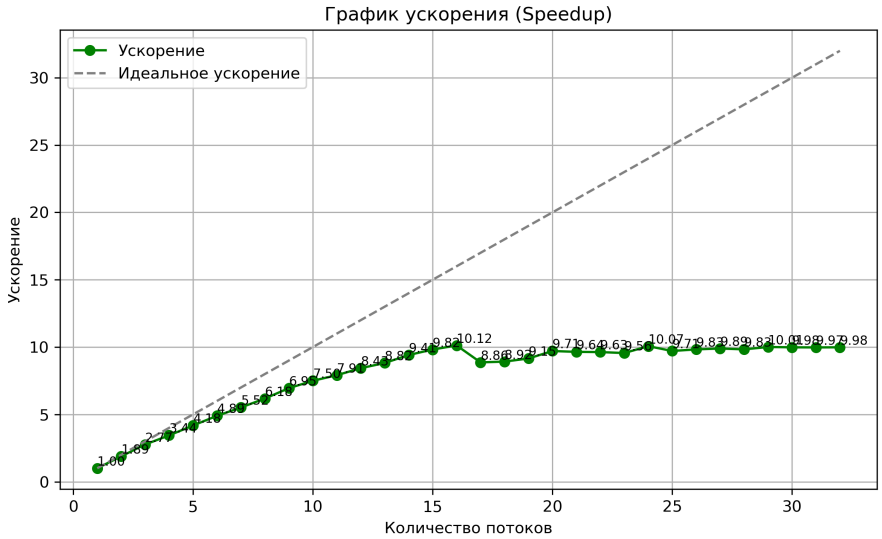

Рис. 2: График зависимости ускорения от количества используемых потоков.

На представленных графиках показаны результаты экспериментального исследования производительности параллельной программы в зависимости от количества используемых потоков. Первый график иллюстрирует зависимость времени выполнения программы от числа потоков: при увеличении числа потоков с одного до примерно восьми–двенадцати

время выполнения резко снижается, что свидетельствует о высокой эффективности распараллеливания на этом этапе; однако начиная с 16–20 потоков кривая выходит на плато — дальнейшее увеличение числа потоков практически не приводит к уменьшению времени выполнения, что обусловлено накладными расходами на управление потоками, ограниченной параллелизуемостью задачи по закону Амдала и аппаратными ограничениями процессора. Второй график демонстрирует зависимость ускорения от числа потоков: реальное ускорение сначала близко к идеальному, но затем отклоняется от него, стабилизируясь на уровне около 10–11, что подтверждает наличие последовательной части программы и указывает на то, что дальнейшее увеличение числа потоков не даёт практической выгоды — оптимальное количество потоков для данной программы составляет 8–12, при котором достигается максимальная производительность без избыточной нагрузки на систему.

Ключевые особенности

1. Программа создаёт исходную матрицу размером $M \times N$ и два результирующих массива для эрозии и наращивания.

2. Для повышения производительности используется многопоточность: работа распределяется между потоками статически по блокам строк.

3. Максимальное количество потоков, работающих одновременно, задаётся пользователем через ключ запуска программы, что позволяет гибко контролировать нагрузку.

4. Программа поддерживает многократное применение фильтров (K раз) на матрицу, обеспечивая возможность исследования разных сценариев нагрузки.

5. Время выполнения каждого запуска сохраняется в CSV-файл `results.csv`, который затем используется для построения графика зависимости времени от числа потоков.

6. Структура проекта разделена на папки `inc/` и `src/`, что облегчает поддержку и масштабирование кода.

Выводы

В ходе выполнения лабораторной работы были приобретены практические навыки в организации многопоточной обработки данных в операционных системах Linux, а также в безопасной работе с разделяемыми ресурсами при распараллеливании вычислений.
Была разработана и отлажена программа на языке C, реализующая многократное применение фильтров эрозии и наращивания на матрице вещественных чисел. Программа использует стандартные средства многопоточности POSIX (`pthread_t`), что обеспечивает корректное выполнение на системах семейства Unix и позволяет гибко контролировать количество потоков через аргумент командной строки.
В результате работы программа запускает указанное пользователем максимальное число потоков, при этом работа распределяется статически: каждый поток обрабатывает отдельный диапазон строк матрицы. Это минимизирует накладные расходы на синхронизацию и предотвращает гонки данных.
Были обработаны возможные ошибки ввода аргументов командной строки и обеспечена корректная инициализация и завершение всех потоков. Экспериментально подтверждена

эффективность параллельных вычислений: время выполнения сокращается с увеличением числа потоков до аппаратного предела (числа логических ядер процессора), что подтверждается построенным графиком зависимости времени от количества потоков.
Результаты работы демонстрируют, что статическое распределение задач между потоками позволяет достичь высокой производительности и масштабируемости алгоритма при обработке больших матриц.

Исходная программа

Файл filterworker.c

```c
#include "filter_worker.h"
#include <float.h>

static inline float min_in_3x3(float **mat, int M, int N, int i, int j) {
    float min_val = FLT_MAX;
    for (int di = -1; di <= 1; di++)
        for (int dj = -1; dj <= 1; dj++) {
            int ni = i + di;
            int nj = j + dj;
            if (ni >= 0 && ni < M && nj >= 0 && nj < N)
                if (mat[ni][nj] < min_val)
                    min_val = mat[ni][nj];
        }
    return min_val;
}

static inline float max_in_3x3(float **mat, int M, int N, int i, int j) {
    float max_val = -FLT_MAX;
    for (int di = -1; di <= 1; di++)
        for (int dj = -1; dj <= 1; dj++) {
            int ni = i + di;
            int nj = j + dj;
            if (ni >= 0 && ni < M && nj >= 0 && nj < N)
                if (mat[ni][nj] > max_val)
                    max_val = mat[ni][nj];
        }
    return max_val;
}

void* apply_filter_static(void* arg) {
    ThreadWork *data = (ThreadWork*)arg;

    for (int rep = 0; rep < data->K; rep++) {
        for (int i = data->start_row; i < data->end_row; i++)
            for (int j = 0; j < data->N; j++) {
                data->erosion_result[i][j] = min_in_3x3(data->input, data->M, data->N,
                    i, j);
                data->dilation_result[i][j] = max_in_3x3(data->input, data->M, data->N,
                     i, j);
            }
    }
    return NULL;
}
```

Листинг 1: Работа с потоками

Файл matrixutils.c

```c
#include "matrix_utils.h"
#include <stdio.h>
#include <time.h>
```

```c

float** create_matrix(int M, int N) {
    float **mat = malloc(M * sizeof(float*));
    for (int i = 0; i < M; i++)
        mat[i] = malloc(N * sizeof(float));
    return mat;
}

void fill_matrix(float** mat, int M, int N) {
    srand(time(NULL));
    for (int i = 0; i < M; i++)
        for (int j = 0; j < N; j++)
            mat[i][j] = (float)(rand()) / RAND_MAX;
}

void free_matrix(float** mat, int M) {
    for (int i = 0; i < M; i++)
        free(mat[i]);
    free(mat);
}
```

Листинг 2: Работа с матрицами

Файл matrixfilters.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include "matrix_utils.h"
#include "filter_worker.h"
#include "timer.h"
#include "matrix_filters.h"

static double run_filters_static(int M, int N, int K, int num_threads) {
    float **mat = create_matrix(M, N);
    float **erosion = create_matrix(M, N);
    float **dilation = create_matrix(M, N);
    fill_matrix(mat, M, N);

    pthread_t *threads = malloc(num_threads * sizeof(pthread_t));
    ThreadWork *thread_data = malloc(num_threads * sizeof(ThreadWork));

    int rows_per_thread = M / num_threads;
    int extra = M % num_threads;
    int row_start = 0;

    double start = get_time_sec();

    for (int t = 0; t < num_threads; t++) {
        int row_end = row_start + rows_per_thread + (t < extra ? 1 : 0);
        thread_data[t].input = mat;
        thread_data[t].erosion_result = erosion;
        thread_data[t].dilation_result = dilation;
        thread_data[t].M = M;
        thread_data[t].N = N;
        thread_data[t].K = K;
```

```
        thread_data[t].start_row = row_start;
        thread_data[t].end_row = row_end;
        pthread_create(&threads[t], NULL, apply_filter_static, &thread_data[t]);
        row_start = row_end;
    }

    for (int t = 0; t < num_threads; t++)
        pthread_join(threads[t], NULL);

    double end = get_time_sec();

    free_matrix(mat, M);
    free_matrix(erosion, M);
    free_matrix(dilation, M);
    free(threads);
    free(thread_data);

    return end - start;
}

int run_matrix_filters(int argc, char *argv[]) {
    if (argc < 5) {
        return 1;
    }

    int M = atoi(argv[1]);
    int N = atoi(argv[2]);
    int K = atoi(argv[3]);
    int Tmax = atoi(argv[4]);

    FILE *f = fopen("results.csv", "w");
    if (!f) {
        perror("Error create results.csv");
        return 1;
    }
    fprintf(f, "threads,time\n");

    for (int threads = 1; threads <= Tmax; threads *= 2) {
        double T = run_filters_static(M, N, K, threads);
        fprintf(f, "%d,%.6f\n", threads, T);
        printf("flow: %d, time: %.6f sec\n", threads, T);
    }

    fclose(f);
    printf("Results in results.csv\n");
    return 0;
}
```

Листинг 3: Обработка ввода, подсчет времени

Файл main.c

```
#include "matrix_utils.h"
#include <stdio.h>
#include <time.h>

```

```
5  float** create_matrix(int M, int N) {
6      float **mat = malloc(M * sizeof(float*));
7      for (int i = 0; i < M; i++)
8          mat[i] = malloc(N * sizeof(float));
9      return mat;
10 }
11
12 void fill_matrix(float** mat, int M, int N) {
13     srand(time(NULL));
14     for (int i = 0; i < M; i++)
15         for (int j = 0; j < N; j++)
16             mat[i][j] = (float)(rand()) / RAND_MAX;
17 }
18
19 void free_matrix(float** mat, int M) {
20     for (int i = 0; i < M; i++)
21         free(mat[i]);
22     free(mat);
23 }
```

Листинг 4: Файл main.c

Вывод strace

```
execve("./matrix_filters", ["./matrix_filters", "1000", "1000", "5", "4"], ...)
brk(NULL)
mmap(NULL, size, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0)
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC)
mmap(...)
clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|...})
futex(0x..., FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, tid, NULL, FUTEX_BITSET_MATCH_AN
openat(AT_FDCWD, "results.csv", O_WRONLY|O_CREAT|O_TRUNC, 0666)
write(3, "threads,time\n1,0.372066\n...", 46)
write(1, "Потоков: 1, время: 0.372066 сек", 47)
close(3)
exit_group(0)
```