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»

Группа: М8О-210Б-23

Студент: Кудаева В.А.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 24.12.24

Москва, 2024

Постановка задачи

Вариант 12:

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

Наложить K раз фильтры эрозии и наращивания на матрицу, состоящую из вещественных чисел. На выходе получается 2 результирующие матрицы.

Общий метод и алгоритм решения

Использованные системные вызовы:

- ssize_t write(int fd, const void *buf, size_t count) – записывает по дескриптору fd count байт из buf;
- int pthread_mutex_lock(pthread_mutex_t *mutex) – блокировка мьютекса;
- int pthread_mutex_unlock(pthread_mutex_t *mutex) – разблокировка мьютекса;
- int pthread_cond_wait(pthread_cond_t *cond, pthread_mutex_t *mutex) - переводит поток в ожидающее состояние, пока не будет получен сигнал с помощью условной переменной;
- int pthread_cond_signal(pthread_cond_t *cond) - отправляет сигнал с помощью условной переменной;
- int pthread_create(pthread_t *thread, const pthread_attr_t *attr, void *(*start_routine) (void *), void *arg) – создание потока;
- int pthread_join(pthread_t thread, void **retval) – ожидание завершения потока;
- void exit(int status) – завершение программы с заданным кодом возврата.

Программа предназначена для выполнения операций эрозии и наращивания над матрицей заданное количество раз. Она работает в многопоточном режиме, используя мьютексы и условные переменные для синхронизации потоков.

В начале каждой итерации вычисляется количество потоков, которые могут быть задействованы. Если число потоков больше количества строк в матрице, оно уменьшается до количества строк, чтобы избежать выделения потоков, которые не будут выполнять полезной работы. Массивы threads и tasks создаются для хранения идентификаторов потоков и данных, которые будут переданы каждому потоку. Затем вычисляется количество строк, которое будет обрабатываться каждым потоком.

Цикл распределяет строки между потоками. Сначала поток ожидает своей очереди, если текущее количество активных потоков достигло лимита. Мьютекс thread_count_mutex блокируется для защиты глобальной переменной active_threads, которая отслеживает количество потоков, выполняющихся в данный момент времени. Она используется для ограничения числа одновременно работающих потоков до значения max_threads. Поток может продолжить выполнение только после получения сигнала об освобождении потока через условную переменную thread_available. Далее заполняется структура Task для текущей итерации. Функция process_task выполняет операцию эрозии и наращивания на указанном диапазоне строк. После запуска всех потоков текущей итерации программа ожидает их завершения. Функция pthread_join блокирует выполнение основного потока до тех пор, пока соответствующий поток не завершит работу. После завершения всех потоков результаты операций эрозии и наращивания копируются обратно в исходные матрицы.

Код программы

```c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <unistd.h>
#include <math.h>
#include <string.h>
#include <time.h>

#define ROWS 5
#define COLS 5

typedef struct {
    int start_row;
    int end_row;
    double **matrix_erosion;
    double **result_erosion;
    double **matrix_dilation;
    double **result_dilation;
} Task;

pthread_mutex_t thread_count_mutex;
pthread_cond_t thread_available;
int active_threads = 0;

void write_message(const char* message) {
    if (write(STDOUT_FILENO, message, strlen(message)) == -1) {
        perror("Error: Error writing message\n");
        exit(EXIT_FAILURE);
    }
}

```

```c
double** allocateMatrix(int rows, int cols) {
    double** matrix = (double**)malloc(rows * sizeof(double*));
    if (matrix == NULL) {
        write_message("Error: Memory allocation failed\n");
        exit(EXIT_FAILURE);
    }
    for (int i = 0; i < rows; i++) {
        matrix[i] = (double*)malloc(cols * sizeof(double));
        if (matrix[i] == NULL) {
            write_message("Error: Memory allocation failed\n");
            exit(EXIT_FAILURE);
        }
    }
    return matrix;
}

void freeMatrix(double** matrix, int rows) {
    for (int i = 0; i < rows; i++) {
        free(matrix[i]);
    }
    free(matrix);
}

```

```c
void generateMatrix(double** matrix, int rows, int cols) {
    unsigned int seed = 12345;
    double min = 0.0, max = 10.0;
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            matrix[i][j] = min + (max - min) * ((double)rand_r(&seed) / (double)RAND_MAX);
        }
    }
}

void writeMatrix(double** matrix, int rows, int cols) {
    char buffer[128];
    int length;
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            length = snprintf(buffer, sizeof(buffer), "%5.1f ", matrix[i][j]);
            if (length < 0) {
                write_message("Error: Failed to format string\n");
                exit(EXIT_FAILURE);
            }
            if (write(STDOUT_FILENO, buffer, length) == -1) {
                write_message("Error: Write failed\n");
                exit(EXIT_FAILURE);
            }
        }
        if (write(STDOUT_FILENO, "\n", 1) == -1) {
            write_message("Error: Write failed\n");
            exit(EXIT_FAILURE);
        }
    }
}
```

```c
void dilation(double** input, double** output, int start_rows, int end_rows) {
    for (int i = start_rows; i < end_rows; i++) {
        for (int j = 0; j < COLS; j++) {
            double maxVal = input[i][j];

            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 < ROWS && nj >= 0 && nj < COLS) {
                        if (input[ni][nj] > maxVal) {
                            maxVal = input[ni][nj];
                        }
                    }
                }
            }
            output[i][j] = maxVal;
        }
    }
}
```

```c
void erosion(double** input, double** output, int start_rows, int end_rows) {
    for (int i = start_rows; i < end_rows; i++) {
        for (int j = 0; j < COLS; j++) {
            double minVal = input[i][j];

            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 < ROWS && nj >= 0 && nj < COLS) {
                        if (input[ni][nj] < minVal) {
                            minVal = input[ni][nj];
                        }
                    }
                }
            }
            output[i][j] = minVal;
        }
    }
}
```

```c
void *process_task(void *arg) {
    Task *task = (Task *)arg;

    erosion(task->matrix_erosion, task->result_erosion, task->start_row, task->end_row);
    dilation(task->matrix_dilation, task->result_dilation, task->start_row, task->end_row);

    pthread_mutex_lock(&thread_count_mutex);
    active_threads--;
    pthread_cond_signal(&thread_available);
    pthread_mutex_unlock(&thread_count_mutex);

    pthread_exit(NULL);
}
```

```c

int main(int argc, char *argv[]) {
    if (argc != 3) {
        write_message("Error: Invalid number of arguments\n");
        exit(EXIT_FAILURE);
    }

    int max_threads = atoi(argv[1]);
    int K = atoi(argv[2]);

    if (max_threads < 1 || K < 1) {
        write_message("Error: max_threads and K must be > 0\n");
        exit(EXIT_FAILURE);
    }

    clock_t start = clock();

```

```c
    pthread_mutex_init(&thread_count_mutex, NULL);
    pthread_cond_init(&thread_available, NULL);

    double** erosion_matrix = allocateMatrix(ROWS, COLS);
    double** erosion_result = allocateMatrix(ROWS, COLS);
    double** dilation_matrix = allocateMatrix(ROWS, COLS);
    double** dilation_result = allocateMatrix(ROWS, COLS);

    generateMatrix(erosion_matrix, ROWS, COLS);

    write_message("Input matrix:\n");
    writeMatrix(erosion_matrix, ROWS, COLS);

    for (int k = 0; k < K; k++) {
        max_threads = (max_threads > ROWS) ? ROWS : max_threads;
        pthread_t threads[max_threads];
        Task tasks[max_threads];
        int rows_per_thread = ROWS / max_threads;
        int task_count = 0;

        for (int i = 0; i < max_threads; i++) {
            pthread_mutex_lock(&thread_count_mutex);
            while (active_threads >= max_threads) {
                pthread_cond_wait(&thread_available, &thread_count_mutex);
            }
            active_threads++;
            pthread_mutex_unlock(&thread_count_mutex);

```

```c
            tasks[task_count].start_row = i * rows_per_thread;
            tasks[task_count].end_row = (i == max_threads - 1) ? ROWS : (i + 1) * rows_per_thread;
            tasks[task_count].matrix_erosion = (k == 0) ? erosion_matrix : erosion_result;
            tasks[task_count].result_erosion = erosion_result;

            tasks[task_count].matrix_dilation = (k == 0) ? erosion_matrix : dilation_result;
            tasks[task_count].result_dilation = dilation_result;

            pthread_create(&threads[task_count], NULL, process_task, &tasks[task_count]);
            task_count++;
        }

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

```

```
203         for (int i = 0; i < ROWS; i++) {
204             for (int j = 0; j < COLS; j++) {
205                 erosion_matrix[i][j] = erosion_result[i][j];
206                 dilation_matrix[i][j] = dilation_result[i][j];
207             }
208         }
209     }
```

```
210 
211     write_message("\nErosion matrix:\n");
212     writeMatrix(erosion_result, ROWS, COLS);
213     write_message("\nDilation matrix:\n");
214     writeMatrix(dilation_result, ROWS, COLS);
215 
216     pthread_mutex_destroy(&thread_count_mutex);
217     pthread_cond_destroy(&thread_available);
218 
219     freeMatrix(erosion_matrix, ROWS);
220     freeMatrix(erosion_result, ROWS);
221     freeMatrix(dilation_matrix, ROWS);
222     freeMatrix(dilation_result, ROWS);
223 
224     clock_t end = clock();
225     char time[64];
226     float seconds = (float)(end - start) / CLOCKS_PER_SEC;
227     snprintf(time, sizeof(time), "%lf\n", seconds);
228     write_message(time);
229 
230     return 0;
231 }
```

Протокол работы программы

Тестирование

```
ksenoox@ksenoox:~/task/LabOS/Lab_2$ ./a.out 2 1
Input matrix:
  4.8   7.1   6.4   9.9   7.6
  8.0   6.2   2.6   2.6   8.4
  5.2   9.3   3.6   4.6   3.5
  0.1   2.0   3.2  10.0   9.2
  1.5   3.7   0.2   9.4   1.3

Erosion matrix:
  4.8   2.6   2.6   2.6   2.6
  4.8   2.6   2.6   2.6   2.6
  0.1   0.1   2.0   2.6   2.6
  0.1   0.1   0.2   0.2   1.3
  0.1   0.1   0.2   0.2   1.3

Dilation matrix:
  8.0   8.0   9.9   9.9   9.9
  9.3   9.3   9.9   9.9   9.9
  9.3   9.3  10.0  10.0  10.0
  9.3   9.3  10.0  10.0  10.0
  3.7   3.7  10.0  10.0  10.0
```

```
ksenoox@ksenoox:~/task/LabOS/Lab_2$ ./a.out 4 5
Input matrix:
  4.8   7.1   6.4   9.9   7.6
  8.0   6.2   2.6   2.6   8.4
  5.2   9.3   3.6   4.6   3.5
  0.1   2.0   3.2  10.0   9.2
  1.5   3.7   0.2   9.4   1.3

Erosion matrix:
  0.1   0.1   0.1   0.1   0.1
  0.1   0.1   0.1   0.1   0.1
  0.1   0.1   0.1   0.1   0.1
  0.1   0.1   0.1   0.1   0.1
  0.1   0.1   0.1   0.1   0.1

Dilation matrix:
 10.0  10.0  10.0  10.0  10.0
 10.0  10.0  10.0  10.0  10.0
 10.0  10.0  10.0  10.0  10.0
 10.0  10.0  10.0  10.0  10.0
 10.0  10.0  10.0  10.0  10.0
```

Число потоков	Время выполнения, мс	Ускорение	Эффективность
1	823	1	1
2	688	1, 19	0.59
3	623	1,32	0.44
4	579	1,42	0.35
5	511	1,61	0.32

Объяснение

Многопоточность позволяет ускорить выполнение программы, так как задачи разделяются между несколькими потоками и выполняются параллельно. При увеличении числа потоков растут накладные расходы на управление потоками, например, создание, синхронизация, переключение контекста, поэтому эффективность снижается. Для максимальной производительности количество потоков должно быть равно количеству физических ядер. Когда количество потоков превышает количество ядер процессора**,** потоки перестают выполняться физически параллельно. Вместо этого процессор вынужден переключать контекст между потоками, что способствует снижению эффективности.

Вывод strace

execve("./a.out", ["./a.out", "4", "1"], 0x7ffebc72c090 /* 74 vars */) = 0

brk(NULL) = 0x55f03d310000

arch_prctl(0x3001 /* ARCH_??? */, 0x7ffe601c7030) = -1 EINVAL (Invalid argument)

mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f6e97335000

newfstatat(AT_FDCWD, "/usr/lib/openfoam/openfoam2312/platforms/linux64GccDPInt32Opt/lib/dummy", {st_mode=S_IFDIR|0755, st_size=4096, ...}, 0) = 0

openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3

newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=27331, ...}, AT_EMPTY_PATH) = 0

mmap(NULL, 27331, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f6e9732e000

close(3) = 0

openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3

read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0P\237\2\0\0\0\0"..., 832) = 832

pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64) = 784

pread64(3, "\4\0\0\0 \0\0\0\5\0\0\0GNU\0\2\0\0\300\4\0\0\0\3\0\0\0\0\0\0\0"..., 48, 848) = 48

pread64(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0I\17\357\204\3$\f\221\2039x\324\224\323\236S"..., 68, 896) = 68

newfstatat(3, "", {st_mode=S_IFREG|0755, st_size=2220400, ...}, AT_EMPTY_PATH) = 0

pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64) = 784

mmap(NULL, 2264656, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f6e97105000

mprotect(0x7f6e9712d000, 2023424, PROT_NONE) = 0

mmap(0x7f6e9712d000, 1658880, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) = 0x7f6e9712d000

mmap(0x7f6e972c2000, 360448, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1bd000) = 0x7f6e972c2000

mmap(0x7f6e9731b000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x215000) = 0x7f6e9731b000

mmap(0x7f6e97321000, 52816, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7f6e97321000

close(3) = 0

mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE| MAP_ANONYMOUS, -1, 0) = 0x7f6e97102000

arch_prctl(ARCH_SET_FS, 0x7f6e97102740) = 0

set_tid_address(0x7f6e97102a10) = 357885

set_robust_list(0x7f6e97102a20, 24) = 0

rseq(0x7f6e971030e0, 0x20, 0, 0x53053053) = 0

mprotect(0x7f6e9731b000, 16384, PROT_READ) = 0

mprotect(0x55f01ae3a000, 4096, PROT_READ) = 0

mprotect(0x7f6e9736f000, 8192, PROT_READ) = 0

prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0

munmap(0x7f6e9732e000, 27331) = 0

clock_gettime(CLOCK_PROCESS_CPUTIME_ID, {tv_sec=0, tv_nsec=16167500}) = 0

getrandom("\x1f\xbb\x0f\x44\x86\x65\x4c\x6c", 8, GRND_NONBLOCK) = 8

brk(NULL) = 0x55f03d310000

brk(0x55f03d331000) = 0x55f03d331000

write(1, "Input matrix:\n", 14Input matrix:

) = 14

write(1, " 4.8 ", 6 4.8) = 6

write(1, " 7.1 ", 6 7.1) = 6

write(1, " 6.4 ", 6 6.4) = 6

write(1, " 9.9 ", 6 9.9) = 6

write(1, " 7.6 ", 6 7.6) = 6

write(1, "\n", 1

) = 1

write(1, " 8.0 ", 6 8.0) = 6

write(1, " 6.2 ", 6 6.2) = 6

write(1, " 2.6 ", 6 2.6) = 6

write(1, " 2.6 ", 6 2.6)   = 6

write(1, " 8.4 ", 6 8.4)   = 6

write(1, "\n", 1

)   = 1

write(1, " 5.2 ", 6 5.2)   = 6

write(1, " 9.3 ", 6 9.3)   = 6

write(1, " 3.6 ", 6 3.6)   = 6

write(1, " 4.6 ", 6 4.6)   = 6

write(1, " 3.5 ", 6 3.5)   = 6

write(1, "\n", 1

)   = 1

write(1, " 0.1 ", 6 0.1)   = 6

write(1, " 2.0 ", 6 2.0)   = 6

write(1, " 3.2 ", 6 3.2)   = 6

write(1, " 10.0 ", 6 10.0)   = 6

write(1, " 9.2 ", 6 9.2)   = 6

write(1, "\n", 1

)   = 1

write(1, " 1.5 ", 6 1.5)   = 6

write(1, " 3.7 ", 6 3.7)   = 6

write(1, " 0.2 ", 6 0.2)   = 6

write(1, " 9.4 ", 6 9.4)   = 6

write(1, " 1.3 ", 6 1.3)   = 6

write(1, "\n", 1

)   = 1

rt_sigaction(SIGRT_1, {sa_handler=0x7f6e97196870, sa_mask=[], sa_flags=SA_RESTORER|SA_ONSTACK|SA_RESTART|SA_SIGINFO, sa_restorer=0x7f6e97147520}, NULL, 8) = 0

rt_sigprocmask(SIG_UNBLOCK, [RTMIN RT_1], NULL, 8) = 0

mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) = 0x7f6e96901000

mprotect(0x7f6e96902000, 8388608, PROT_READ|PROT_WRITE) = 0

rt_sigprocmask(SIG_BLOCK, ~[], [], 8) = 0

clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, child_tid=0x7f6e97101910, parent_tid=0x7f6e97101910, exit_signal=0, stack=0x7f6e96901000, stack_size=0x7fff00, tls=0x7f6e97101640} => {parent_tid=[357886]}, 88) = 357886

rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0

mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) = 0x7f6e96100000

mprotect(0x7f6e96101000, 8388608, PROT_READ|PROT_WRITE) = 0

futex(0x7f6e97371a98, FUTEX_WAIT_PRIVATE, 2, NULL) = 0

futex(0x7f6e97371a98, FUTEX_WAKE_PRIVATE, 1) = 0

rt_sigprocmask(SIG_BLOCK, ~[], [], 8) = 0

clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, child_tid=0x7f6e96900910, parent_tid=0x7f6e96900910, exit_signal=0, stack=0x7f6e96100000, stack_size=0x7fff00, tls=0x7f6e96900640} => {parent_tid=[357887]}, 88) = 357887

rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0

mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) = 0x7f6e958df000

mprotect(0x7f6e958e0000, 8388608, PROT_READ|PROT_WRITE) = 0

rt_sigprocmask(SIG_BLOCK, ~[], [], 8) = 0

clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, child_tid=0x7f6e960df910, parent_tid=0x7f6e960df910, exit_signal=0, stack=0x7f6e958df000, stack_size=0x7fff00, tls=0x7f6e960df640} => {parent_tid=[0]}, 88) = 357888

rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0

mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) = 0x7f6e950de000

mprotect(0x7f6e950df000, 8388608, PROT_READ|PROT_WRITE) = 0

rt_sigprocmask(SIG_BLOCK, ~[], [], 8) = 0

clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, child_tid=0x7f6e958de910, parent_tid=0x7f6e958de910, exit_signal=0, stack=0x7f6e950de000, stack_size=0x7fff00, tls=0x7f6e958de640} => {parent_tid=[357889]}, 88) = 357889

rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0

write(1, "\nErosion matrix:\n", 17

Erosion matrix:

) = 17

write(1, " 4.8 ", 6 4.8) = 6

write(1, " 2.6 ", 6 2.6) = 6

write(1, " 2.6 ", 6 2.6) = 6

write(1, " 2.6 ", 6 2.6) = 6

write(1, " 2.6 ", 6 2.6) = 6

write(1, "\n", 1

) = 1

write(1, " 4.8 ", 6 4.8) = 6

write(1, " 2.6 ", 6 2.6) = 6

write(1, " 2.6 ", 6 2.6) = 6

write(1, " 2.6 ", 6 2.6) = 6

write(1, " 2.6 ", 6 2.6) = 6

write(1, "\n", 1

) = 1

write(1, " 0.1 ", 6 0.1) = 6

```
write(1, "  0.1 ", 6  0.1 )              = 6
write(1, "  2.0 ", 6  2.0 )              = 6
write(1, "  2.6 ", 6  2.6 )              = 6
write(1, "  2.6 ", 6  2.6 )              = 6
write(1, "\n", 1
)                   = 1
write(1, "  0.1 ", 6  0.1 )              = 6
write(1, "  0.1 ", 6  0.1 )              = 6
write(1, "  0.2 ", 6  0.2 )              = 6
write(1, "  0.2 ", 6  0.2 )              = 6
write(1, "  1.3 ", 6  1.3 )              = 6
write(1, "\n", 1
)                   = 1
write(1, "  0.1 ", 6  0.1 )              = 6
write(1, "  0.1 ", 6  0.1 )              = 6
write(1, "  0.2 ", 6  0.2 )              = 6
write(1, "  0.2 ", 6  0.2 )              = 6
write(1, "  1.3 ", 6  1.3 )              = 6
write(1, "\n", 1
)                   = 1
write(1, "\nDilation matrix:\n", 18
Dilation matrix:
)    = 18
write(1, "  8.0 ", 6  8.0 )              = 6
write(1, "  8.0 ", 6  8.0 )              = 6
write(1, "  9.9 ", 6  9.9 )              = 6
write(1, "  9.9 ", 6  9.9 )              = 6
```

```
write(1, "  9.9 ", 6  9.9 )                = 6
write(1, "\n", 1
)                    = 1
write(1, "  9.3 ", 6  9.3 )                = 6
write(1, "  9.3 ", 6  9.3 )                = 6
write(1, "  9.9 ", 6  9.9 )                = 6
write(1, "  9.9 ", 6  9.9 )                = 6
write(1, "  9.9 ", 6  9.9 )                = 6
write(1, "\n", 1
)                    = 1
write(1, "  9.3 ", 6  9.3 )                = 6
write(1, "  9.3 ", 6  9.3 )                = 6
write(1, " 10.0 ", 6 10.0 )                 = 6
write(1, " 10.0 ", 6 10.0 )                 = 6
write(1, " 10.0 ", 6 10.0 )                 = 6
write(1, "\n", 1
)                    = 1
write(1, "  9.3 ", 6  9.3 )                = 6
write(1, "  9.3 ", 6  9.3 )                = 6
write(1, " 10.0 ", 6 10.0 )                 = 6
write(1, " 10.0 ", 6 10.0 )                 = 6
write(1, " 10.0 ", 6 10.0 )                 = 6
write(1, "\n", 1
)                    = 1
write(1, "  3.7 ", 6  3.7 )                = 6
write(1, "  3.7 ", 6  3.7 )                = 6
write(1, " 10.0 ", 6 10.0 )                 = 6
```

```
write(1, " 10.0 ", 6 10.0 )                = 6
write(1, " 10.0 ", 6 10.0 )                = 6
write(1, "\n", 1
)                       = 1
clock_gettime(CLOCK_PROCESS_CPUTIME_ID, {tv_sec=0, tv_nsec=24735000}) = 0
write(1, "0.008568\n", 90.008568
)              = 9
exit_group(0)                           = ?
+++ exited with 0 +++
```

Вывод

В ходе выполнения лабораторной работы мне удалось написать многопоточную программу для выполнения операций эрозии и наращивания матрицы с использованием синхронизации потоков. Программа была протестирована на различных количествах потоков, и получены экспериментальные данные о времени выполнения, ускорении и эффективности.