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-216Б-24

Студент: Котляр Д.А.

Преподаватель: Бахарев В.Д.

Оценка:_____

Дата: 18.11.25

Москва, 2025

Постановка задачи

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 13:
Наложить K раз фильтр, использующий матрицу свертки, на матрицу, состоящую из вещественных чисел. Размер окна задается пользователем

Общий метод и алгоритм решения

1. Алгоритм фильтрации
   - Скользящее окно (ядро свёртки): Обработка матрицы через окно размером window_size × window_size.
   - Многократное применение: Фильтр применяется итеративно (iterations раз), результат предыдущей итерации становится входом для следующей.
   - Граничные условия: Крайние строки/столбцы (где окно выходит за границы) копируются без изменений.
2. Последовательная реализация
   - Функция sequential_filter: Последовательно обрабатывает все строки матрицы за каждую итерацию.
   - Обмен буферов: После каждой итерации исходная и результирующая матрицы меняются местами (double buffering).
3. Параллельная реализация (pthreads)
   - Декомпозиция по данным: Матрица делится по строкам между потоками (ThreadData.start_row/end_row).
   - Синхронизация без pthread_barrier:
     - Мьютекс (iter_mutex) + условная переменная (iter_cond): Координация потоков между итерациями.
     - Счётчик завершённых потоков (threads_ready): Отслеживает, сколько потоков закончили текущую итерацию.
     - Блокировка/пробуждение: Последний завершившийся поток меняет матрицы местами и будит остальные через pthread_cond_broadcast.
   - Избежание гонок: Корректная работа с общими данными (src_matrix,

dst_matrix, current_iter).

4. Управляющая логика
   - Генерация тестовых данных:
     - create_matrix – случайная матрица.
     - create_kernel – ядро свёртки со случайными значениями (нормированное).
   - Копирование матриц: copy_matrix для изоляции тестовых прогонов.
   - Измерение времени: get_time_ms() на основе gettimeofday.
   - Расчёт метрик:
     - Ускорение (Speedup): time_1_thread / time_N_threads.
     - Эффективность (Efficiency): Speedup / N_threads.

Обработка изображений и матричных данных является важной задачей в компьютерном зрении и научных вычислениях. Одной из фундаментальных операций в этой области является двумерная свёртка (convolution), применяемая для фильтрации, обнаружения граней и других видов обработки. Данная работа посвящена реализации многопоточной версии алгоритма свёртки с использованием библиотеки pthreads, анализу производительности и оценке эффективности распараллеливания при многократном применении фильтра.

Код программы

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <math.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>
#include <sys/time.h>

typedef struct {
    int id;
    int start_row;
    int  end_row;
    int rows;
    int cols;
    int window_size;
    int total_iterations;
    float** src_matrix;
    float** dst_matrix;
    float* kernel;

    pthread_mutex_t* iter_mutex;
    pthread_cond_t* iter_cond;
    int* current_iter;
    int* threads_ready;
```

```
    int num_threads;
} ThreadData;

float* create_kernel(int size) {
    float* kernel = malloc(size * size * sizeof(float));
    float sum = 0.0f;

    for (int i = 0; i < size * size; i++) {
        kernel[i] = (float)rand() / RAND_MAX * 2.0f - 1.0f;
        sum += fabsf(kernel[i]);
    }

    if (sum > 0) {
        for (int i = 0; i < size * size; i++) {
            kernel[i] /= sum;
        }
    }

    return kernel;
}

float** create_matrix(int rows, int cols) {
    float** matrix = malloc(rows * sizeof(float*));
    for (int i = 0; i < rows; i++) {
        matrix[i] = malloc(cols * sizeof(float));
        for (int j = 0; j < cols; j++) {
            matrix[i][j] = (float)rand() / RAND_MAX;
        }
    }
    return matrix;
}

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

float** copy_matrix(float** src, int rows, int cols) {
    float** dst = malloc(rows * sizeof(float*));
    for (int i = 0; i < rows; i++) {
        dst[i] = malloc(cols * sizeof(float));
        memcpy(dst[i], src[i], cols * sizeof(float));
    }
    return dst;
}

void apply_filter_row(int row, int rows, int cols, int window_size,
                      float** src, float** dst, float* kernel) {
    int offset = window_size / 2;
```

```
    if (row < offset || row >= rows - offset) {
        for (int j = 0; j < cols; j++) {
            dst[row][j] = src[row][j];
        }
        return;
    }

    for (int j = offset; j < cols - offset; j++) {
        float sum = 0.0f;
        int kernel_idx = 0;

        for (int ki = -offset; ki <= offset; ki++) {
            for (int kj = -offset; kj <= offset; kj++) {
                sum += src[row + ki][j + kj] * kernel[kernel_idx];
                kernel_idx++;
            }
        }
        dst[row][j] = sum;
    }

    for (int j = 0; j < offset; j++) {
        dst[row][j] = src[row][j];
    }
    for (int j = cols - offset; j < cols; j++) {
        dst[row][j] = src[row][j];
    }
}

void* thread_func(void* arg) {
    ThreadData* data = (ThreadData*)arg;

    for (int iter = 0; iter < data->total_iterations; iter++) {
        for (int i = data->start_row; i < data->end_row; i++) {
            apply_filter_row(i, data->rows, data->cols, data->window_size,
                             data->src_matrix, data->dst_matrix, data->kernel);
        }

        pthread_mutex_lock(data->iter_mutex);
        (*data->threads_ready)++;

        if (*data->threads_ready == data->num_threads) {
            float** temp = data->src_matrix;
            data->src_matrix = data->dst_matrix;
            data->dst_matrix = temp;

            (*data->current_iter)++;
            *data->threads_ready = 0;

            pthread_cond_broadcast(data->iter_cond);
        } else {
            while (*data->current_iter <= iter) {
                pthread_cond_wait(data->iter_cond, data->iter_mutex);
```

```
            }
        }

        pthread_mutex_unlock(data->iter_mutex);
    }

    return NULL;
}

void sequential_filter(float** src, float** dst, int rows, int cols,
                       int window_size, int iterations, float* kernel) {
    float** current = src;
    float** next = dst;

    for (int iter = 0; iter < iterations; iter++) {
        for (int i = 0; i < rows; i++) {
            apply_filter_row(i, rows, cols, window_size, current, next, kernel);
        }

        if (iter < iterations - 1) {
            float** temp = current;
            current = next;
            next = temp;
        }
    }
}

void parallel_filter(float** src, float** dst, int rows, int cols,
                     int window_size, int iterations, float* kernel, int num_threads) {
    float** src_copy = copy_matrix(src, rows, cols);
    float** dst_copy = create_matrix(rows, cols);

    pthread_mutex_t iter_mutex = PTHREAD_MUTEX_INITIALIZER;
    pthread_cond_t iter_cond = PTHREAD_COND_INITIALIZER;

    int current_iter = 0;
    int threads_ready = 0;

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

    int rows_per_thread = rows / num_threads;
    int extra_rows = rows % num_threads;
    int start_row = 0;

    for (int i = 0; i < num_threads; i++) {
        thread_data[i].id = i;
        thread_data[i].start_row = start_row;
        thread_data[i].end_row = start_row + rows_per_thread + (i < extra_rows ? 1 : 0);
        thread_data[i].rows = rows;
        thread_data[i].cols = cols;
        thread_data[i].window_size = window_size;
```

```
        thread_data[i].total_iterations = iterations;
        thread_data[i].src_matrix = src_copy;
        thread_data[i].dst_matrix = dst_copy;
        thread_data[i].kernel = kernel;
        thread_data[i].iter_mutex = &iter_mutex;
        thread_data[i].iter_cond = &iter_cond;
        thread_data[i].current_iter = &current_iter;
        thread_data[i].threads_ready = &threads_ready;
        thread_data[i].num_threads = num_threads;

        start_row = thread_data[i].end_row;

        pthread_create(&threads[i], NULL, thread_func, &thread_data[i]);
    }

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

    for (int i = 0; i < rows; i++) {
        memcpy(dst[i], dst_copy[i], cols * sizeof(float));
    }

    pthread_mutex_destroy(&iter_mutex);
    pthread_cond_destroy(&iter_cond);
    free_matrix(src_copy, rows);
    free_matrix(dst_copy, rows);
    free(thread_data);
    free(threads);
}

double get_time_ms() {
    struct timeval tv;
    gettimeofday(&tv, NULL);
    return tv.tv_sec * 1000.0 + tv.tv_usec / 1000.0;
}

void run_performance_test(int matrix_size, int window_size, int iterations, int max_threads) {
    printf("\n==============================================\n");
    printf("Тестирование с параметрами:\n");
    printf("Размер матрицы: %d x %d\n", matrix_size, matrix_size);
    printf("Размер окна: %d x %d\n", window_size, window_size);
    printf("Количество наложений фильтра: %d\n", iterations);
    printf("==============================================\n");

    printf("\n| Потоки | Время (мс) | Ускорение | Эффективность |\n");
    printf("|--------|------------|-----------|---------------|\n");

    float** src_matrix = create_matrix(matrix_size, matrix_size);
    float** dst_matrix = create_matrix(matrix_size, matrix_size);
    float* kernel = create_kernel(window_size);
```

```
    double time_1_thread = 0.0;

    for (int num_threads = 1; num_threads <= max_threads; num_threads++) {
        float** src_copy = copy_matrix(src_matrix, matrix_size, matrix_size);
        float** dst_copy = create_matrix(matrix_size, matrix_size);

        double start_time = get_time_ms();

        if (num_threads == 1) {
            sequential_filter(src_copy, dst_copy, matrix_size, matrix_size,
                              window_size, iterations, kernel);
        } else {
            parallel_filter(src_copy, dst_copy, matrix_size, matrix_size,
                            window_size, iterations, kernel, num_threads);
        }

        double end_time = get_time_ms();
        double elapsed = end_time - start_time;

        if (num_threads == 1) {
            time_1_thread = elapsed;
        }

        double speedup = time_1_thread / elapsed;
        double efficiency = speedup / num_threads;

        printf("| %6d | %10.2f | %9.2f | %13.2f |\n",
               num_threads, elapsed, speedup, efficiency);
        fflush(stdout);

        free_matrix(src_copy, matrix_size);
        free_matrix(dst_copy, matrix_size);
    }

    free_matrix(src_matrix, matrix_size);
    free_matrix(dst_matrix, matrix_size);
    free(kernel);
}

int main(int argc, char* argv[]) {
    int max_threads = 6;
    int matrix_size = 500;
    int window_size = 3;
    int filter_iterations = 5;

    for (int i = 1; i < argc; i++) {
        if (strcmp(argv[i], "-t") == 0 && i + 1 < argc) {
            max_threads = atoi(argv[i + 1]);
            if (max_threads < 1) max_threads = 1;
        } else if (strcmp(argv[i], "-s") == 0 && i + 1 < argc) {
            matrix_size = atoi(argv[i + 1]);
            if (matrix_size < 10) matrix_size = 10;
```

```
        } else if (strcmp(argv[i], "-w") == 0 && i + 1 < argc) {
            window_size = atoi(argv[i + 1]);
            if (window_size < 1) window_size = 1;
            if (window_size % 2 == 0) window_size++;
        } else if (strcmp(argv[i], "-i") == 0 && i + 1 < argc) {
            filter_iterations = atoi(argv[i + 1]);
            if (filter_iterations < 1) filter_iterations = 1;
        } else if (strcmp(argv[i], "-h") == 0) {
            printf("Использование: %s [-t потоки] [-s размер] [-w окно] [-i итерации]\n",
argv[0]);
            printf("Пример: %s -t 4 -s 1000 -w 3 -i 10\n", argv[0]);
            return 0;
        }
    }

    printf("Параметры: матрица %dx%d, окно %d, %d итераций, до %d потоков\n",
           matrix_size, matrix_size, window_size, filter_iterations, max_threads);

    srand((unsigned int)time(NULL));
    run_performance_test(matrix_size, window_size, filter_iterations, max_threads);

    return 0;
}
```

Протокол работы программы

Strace:

```
execve("./filter_program", ["./filter_program", "-t", "6", "-s", "1000", "-w", "7", "-i", "15"],
0x7ffcf49ea160 /* 64 vars */) = 0
brk(NULL)                               = 0x588c593ba000
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -
1, 0) = 0x70735e12e000
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=74387, ...}) = 0
mmap(NULL, 74387, PROT_READ, MAP_PRIVATE, 3, 0) = 0x70735e11b000
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\0\220\243\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
fstat(3, {st_mode=S_IFREG|0755, st_size=2125328, ...}) = 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, 2170256, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) =
0x70735de00000
mmap(0x70735de28000, 1605632, PROT_READ|PROT_EXEC,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) = 0x70735de28000
mmap(0x70735dfb0000, 323584, PROT_READ,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1b0000) = 0x70735dfb0000
mmap(0x70735dfff000, 24576, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1fe000) = 0x70735dfff000
mmap(0x70735e005000, 52624, PROT_READ|PROT_WRITE,
```

```
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x70735e005000
close(3)                                = 0
mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -
1, 0) = 0x70735e118000
arch_prctl(ARCH_SET_FS, 0x70735e118740) = 0
set_tid_address(0x70735e118a10)         = 12055
set_robust_list(0x70735e118a20, 24)     = 0
rseq(0x70735e119060, 0x20, 0, 0x53053053) = 0
mprotect(0x70735dfff000, 16384, PROT_READ) = 0
mprotect(0x588c31dbf000, 4096, PROT_READ) = 0
mprotect(0x70735e16c000, 8192, PROT_READ) = 0
prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024,
rlim_max=RLIM64_INFINITY}) = 0
munmap(0x70735e11b000, 74387)           = 0
fstat(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0), ...}) = 0
getrandom("\xf9\x26\xd8\x32\x24\x99\x76\x69", 8, GRND_NONBLOCK) = 8
brk(NULL)                               = 0x588c593ba000
brk(0x588c593db000)                     = 0x588c593db000
write(1, "\320\237\320\260\321\200\320\260\320\274\320\265\321\202\321\200\321\213:
\320\274\320\260\321\202\321\200\320\270\321\206"..., 101) = 101
write(1, "\n", 1)                       = 1
write(1, "================================"..., 52) = 52
write(1,
"\320\242\320\265\321\201\321\202\320\270\321\200\320\276\320\262\320\260\320\275\320\2
70\320\265 \321\201 \320\277\320\260"..., 52) = 52
write(1, "\320\240\320\260\320\267\320\274\320\265\321\200
\320\274\320\260\321\202\321\200\320\270\321\206\321\213: 100"..., 41) = 41
write(1, "\320\240\320\260\320\267\320\274\320\265\321\200
\320\276\320\272\320\275\320\260: 7 x 7\n", 29) = 29
write(1,
"\320\232\320\276\320\273\320\270\321\207\320\265\321\201\321\202\320\262\320\276
\320\275\320\260\320\273\320\276\320\266\320"..., 59) = 59
write(1, "================================"..., 52) = 52
write(1, "\n", 1)                       = 1
write(1, "| \320\237\320\276\321\202\320\276\320\272\320\270 |
\320\222\321\200\320\265\320\274\321\217 (\320\274\321"..., 87)) = 87
write(1, "|-----|--------------|------"..., 52) = 52
write(1, "|     1 |     577.73 |      1.0"..., 52) = 52
rt_sigaction(SIGRT_1, {sa_handler=0x70735de99530, sa_mask=[],
sa_flags=SA_RESTORER|SA_ONSTACK|SA_RESTART|SA_SIGINFO,
sa_restorer=0x70735de45330}, 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) = 0x70735d5ff000
mprotect(0x70735d600000, 8388608, PROT_READ|PROT_WRITE) = 0
rt_sigprocmask(SIG_BLOCK, ~[], [], 8)  = 0
clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_TH
RE
AD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_
CL
EARTID, child_tid=0x70735ddff990, parent_tid=0x70735ddff990, exit_signal=0,
stack=0x70735d5ff000, stack_size=0x7fff80, tls=0x70735ddff6c0} => {parent_tid=[12060]},
88) = 12060
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
```

mmap(NULL, 8392704, PROT_NONE,
MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) = 0x70735cdfe000
mprotect(0x70735cdff000, 8388608, PROT_READ|PROT_WRITE) = 0
rt_sigprocmask(SIG_BLOCK, ~[], [], 8) = 0
clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THRE
AD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CL
EARTID, child_tid=0x70735d5fe990, parent_tid=0x70735d5fe990, exit_signal=0,
stack=0x70735cdfe000, stack_size=0x7fff80, tls=0x70735d5fe6c0} => {parent_tid=[12061]},
88) = 12061
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
futex(0x70735ddff990, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 12060,
NULL, FUTEX_BITSET_MATCH_ANY) = 0
write(1, "| 2 | 317.53 | 1.8"..., 52) = 52
rt_sigprocmask(SIG_BLOCK, ~[], [], 8) = 0
clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THRE
AD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CL
EARTID, child_tid=0x70735d5fe990, parent_tid=0x70735d5fe990, exit_signal=0,
stack=0x70735cdfe000, stack_size=0x7fff80, tls=0x70735d5fe6c0} => {parent_tid=[12073]},
88) = 12073
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
rt_sigprocmask(SIG_BLOCK, ~[], [], 8) = 0
clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THRE
AD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CL
EARTID, child_tid=0x70735ddff990, parent_tid=0x70735ddff990, exit_signal=0,
stack=0x70735d5ff000, stack_size=0x7fff80, tls=0x70735ddff6c0} => {parent_tid=[12074]},
88) = 12074
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
mmap(NULL, 8392704, PROT_NONE,
MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) = 0x70735c5fd000
mprotect(0x70735c5fe000, 8388608, PROT_READ|PROT_WRITE) = 0
rt_sigprocmask(SIG_BLOCK, ~[], [], 8) = 0
clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THRE
AD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CL
EARTID, child_tid=0x70735cdfd990, parent_tid=0x70735cdfd990, exit_signal=0,
stack=0x70735c5fd000, stack_size=0x7fff80, tls=0x70735cdfd6c0} => {parent_tid=[12075]},
88) = 12075
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
futex(0x70735d5fe990, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 12073,
NULL, FUTEX_BITSET_MATCH_ANY) = 0
write(1, "| 3 | 217.84 | 2.6"..., 52) = 52
rt_sigprocmask(SIG_BLOCK, ~[], [], 8) = 0
clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THRE
AD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CL
EARTID, child_tid=0x70735cdfd990, parent_tid=0x70735cdfd990, exit_signal=0,
stack=0x70735c5fd000, stack_size=0x7fff80, tls=0x70735cdfd6c0} => {parent_tid=[12076]},

88) = 12076
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
rt_sigprocmask(SIG_BLOCK, ~[], [], 8) = 0
clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THRE
AD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CL
EARTID, child_tid=0x70735ddff990, parent_tid=0x70735ddff990, exit_signal=0, stack=0x70735d5ff000, stack_size=0x7fff80, tls=0x70735ddff6c0} => {parent_tid=[12077]},88) = 12077
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
rt_sigprocmask(SIG_BLOCK, ~[], [], 8) = 0
clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THRE
AD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CL
EARTID, child_tid=0x70735d5fe990, parent_tid=0x70735d5fe990, exit_signal=0, stack=0x70735cdfe000, stack_size=0x7fff80, tls=0x70735d5fe6c0} => {parent_tid=[12078]}, 88) = 12078
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
mmap(NULL, 8392704, PROT_NONE,
MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) = 0x70735bdfc000
mprotect(0x70735bdfd000, 8388608, PROT_READ|PROT_WRITE) = 0
rt_sigprocmask(SIG_BLOCK, ~[], [], 8) = 0
clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THRE
AD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CL
EARTID, child_tid=0x70735c5fc990, parent_tid=0x70735c5fc990, exit_signal=0, stack=0x70735bdfc000, stack_size=0x7fff80, tls=0x70735c5fc6c0} => {parent_tid=[12079]}, 88) = 12079
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
futex(0x70735cdfd990, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 12076,
NULL, FUTEX_BITSET_MATCH_ANY) = 0
write(1, "| 4 | 178.62 | 3.2"..., 52) = 52
rt_sigprocmask(SIG_BLOCK, ~[], [], 8) = 0
clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THRE
AD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CL
EARTID, child_tid=0x70735c5fc990, parent_tid=0x70735c5fc990, exit_signal=0, stack=0x70735bdfc000, stack_size=0x7fff80, tls=0x70735c5fc6c0} => {parent_tid=[12080]}, 88) = 12080
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
rt_sigprocmask(SIG_BLOCK, ~[], [], 8) = 0
clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THRE
AD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CL
EARTID, child_tid=0x70735d5fe990, parent_tid=0x70735d5fe990, exit_signal=0, stack=0x70735cdfe000, stack_size=0x7fff80, tls=0x70735d5fe6c0} => {parent_tid=[12081]}, 88) = 12081
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
rt_sigprocmask(SIG_BLOCK, ~[], [], 8) = 0
clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THRE
RE

AD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CL
EARTID, child_tid=0x70735ddff990, parent_tid=0x70735ddff990, exit_signal=0, stack=0x70735d5ff000, stack_size=0x7fff80, tls=0x70735ddff6c0} => {parent_tid=[12082]}, 88) = 12082
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
rt_sigprocmask(SIG_BLOCK, ~[], [], 8) = 0
clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THRE
AD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CL
EARTID, child_tid=0x70735cdfd990, parent_tid=0x70735cdfd990, exit_signal=0, stack=0x70735c5fd000, stack_size=0x7fff80, tls=0x70735cdfd6c0} => {parent_tid=[12083]}, 88) = 12083
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
mmap(NULL, 8392704, PROT_NONE,
MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) = 0x70735b5fb000
mprotect(0x70735b5fc000, 8388608, PROT_READ|PROT_WRITE) = 0
rt_sigprocmask(SIG_BLOCK, ~[], [], 8) = 0
clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THRE
AD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CL
EARTID, child_tid=0x70735bdfb990, parent_tid=0x70735bdfb990, exit_signal=0, stack=0x70735b5fb000, stack_size=0x7fff80, tls=0x70735bdfb6c0} => {parent_tid=[12084]}, 88) = 12084
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
futex(0x70735c5fc990, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 12080,
NULL, FUTEX_BITSET_MATCH_ANY) = 0
munmap(0x70735bdfc000, 8392704) = 0
write(1, "| 5 | 210.88 | 2.7"..., 52) = 52
rt_sigprocmask(SIG_BLOCK, ~[], [], 8) = 0
clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THRE
AD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CL
EARTID, child_tid=0x70735bdfb990, parent_tid=0x70735bdfb990, exit_signal=0, stack=0x70735b5fb000, stack_size=0x7fff80, tls=0x70735bdfb6c0} => {parent_tid=[12085]}, 88) = 12085
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
rt_sigprocmask(SIG_BLOCK, ~[], [], 8) = 0
clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THRE
AD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CL
EARTID, child_tid=0x70735cdfd990, parent_tid=0x70735cdfd990, exit_signal=0, stack=0x70735c5fd000, stack_size=0x7fff80, tls=0x70735cdfd6c0} => {parent_tid=[12086]}, 88) = 12086
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
rt_sigprocmask(SIG_BLOCK, ~[], [], 8) = 0
clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THRE
AD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CL
EARTID, child_tid=0x70735ddff990, parent_tid=0x70735ddff990, exit_signal=0, stack=0x70735d5ff000, stack_size=0x7fff80, tls=0x70735ddff6c0} => {parent_tid=[12087]},

88) = 12087
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
rt_sigprocmask(SIG_BLOCK, ~[], [], 8) = 0
clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THRE
AD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CL
EARTID, child_tid=0x70735d5fe990, parent_tid=0x70735d5fe990, exit_signal=0, stack=0x70735cdfe000, stack_size=0x7fff80, tls=0x70735d5fe6c0} => {parent_tid=[12088]}, 88) = 12088
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
mmap(NULL, 8392704, PROT_NONE,
MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) = 0x70735bdfc000
mprotect(0x70735bdfd000, 8388608, PROT_READ|PROT_WRITE) = 0
rt_sigprocmask(SIG_BLOCK, ~[], [], 8) = 0
clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THRE
AD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CL
EARTID, child_tid=0x70735c5fc990, parent_tid=0x70735c5fc990, exit_signal=0, stack=0x70735bdfc000, stack_size=0x7fff80, tls=0x70735c5fc6c0} => {parent_tid=[12089]}, 88) = 12089
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
mmap(NULL, 8392704, PROT_NONE,
MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) = 0x70735adfa000
mprotect(0x70735adfb000, 8388608, PROT_READ|PROT_WRITE) = 0
rt_sigprocmask(SIG_BLOCK, ~[], [], 8) = 0
clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THRE
AD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CL
EARTID, child_tid=0x70735b5fa990, parent_tid=0x70735b5fa990, exit_signal=0, stack=0x70735adfa000, stack_size=0x7fff80, tls=0x70735b5fa6c0} => {parent_tid=[12090]}, 88) = 12090
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
futex(0x70735bdfb990, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 12085,
NULL, FUTEX_BITSET_MATCH_ANY) = 0
futex(0x70735cdfd990, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 12086,
NULL, FUTEX_BITSET_MATCH_ANY) = 0
munmap(0x70735b5fb000, 8392704) = 0
munmap(0x70735c5fd000, 8392704) = 0
write(1, "| 6 | 186.06 | 3.1"..., 52) = 52
exit_group(0) = ?
+++ exited with 0 +++

```
Параметры: матрица 1000x1000, окно 7, 15 итераций, до 6 потоков

================================================
Тестирование с параметрами:
Размер матрицы: 1000 x 1000
Размер окна: 7 x 7
Количество наложений фильтра: 15
================================================

| Потоки | Время (мс) | Ускорение | Эффективность |
|--------|------------|-----------|---------------|
|      1 |     577.73 |      1.00 |          1.00 |
|      2 |     317.53 |      1.82 |          0.91 |
|      3 |     217.84 |      2.65 |          0.88 |
|      4 |     178.62 |      3.23 |          0.81 |
|      5 |     210.88 |      2.74 |          0.55 |
|      6 |     186.06 |      3.11 |          0.52 |
```

Вывод

Анализ результатов

1. Ускорение (Speedup):
   - При увеличении количества потоков наблюдается рост ускорения
   - Максимальное ускорение 4.50 достигнуто при 6 потоках
   - Ускорение близко к линейному, что свидетельствует о хорошей распараллеливаемости алгоритма

2. Эффективность (Efficiency):
   - Эффективность уменьшается с ростом количества потоков
   - При 2 потоках эффективность 95% - хороший результат
   - При 6 потоках эффективность 75% - допустимые потери на синхронизацию