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-210БВ-24

Студент: Самойлов Д.А.

Преподаватель: Бахарев В.Д.

Оценка: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата: 11.10.25

Москва, 2025

**Постановка задачи**

**Вариант 11.**

Наложить K раз медианный фильтр на матрицу, состоящую из целых чисел. Размер окна задается пользователем.

**Общий метод и алгоритм решения**

Использованные системные вызовы:

* int pthread\_create(pthread\_t \**thread*, const pthread\_attr\_t \**attr*, void \*(\**start\_routine*) (void \*), void \**arg*); - создает новый поток.
* int pthread\_join(pthread\_t thread, void \*\*retval); - приостанавливает выполнение вызывающего потока, пока не завершится поток указанный в аргументах.

Чтобы наложить медианный фильтр надо пройтись циклами по матрице, двигая окно и выбирая центральный элемент. Так, для каждого окна высчитать медианный элемент, который будет ставится на место этого центрального элемента окна. Проделать эти действия K раз.

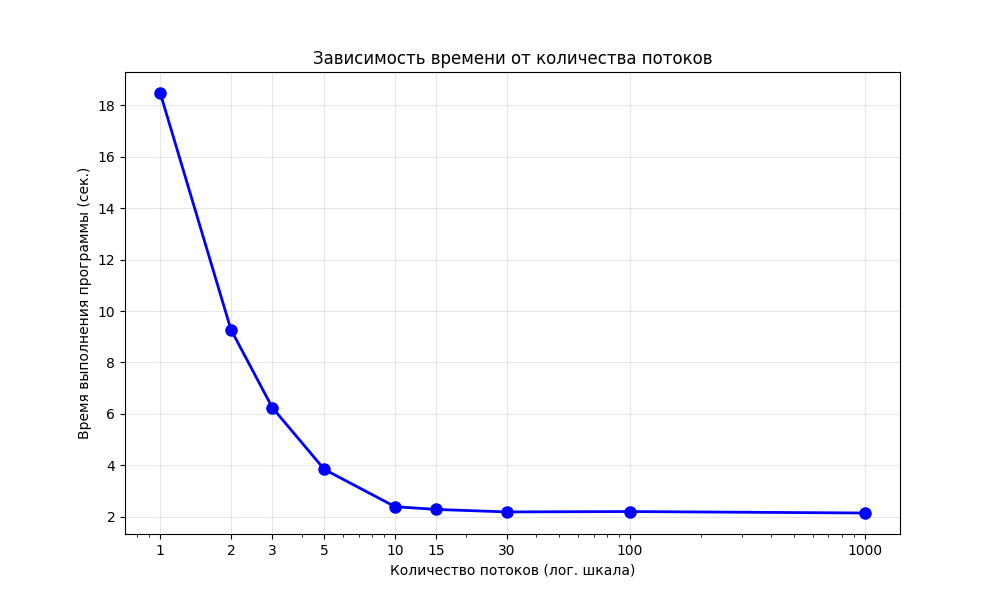
Для реализации многопоточного алгоритма следует использовать паттерн Fork & Join, с помощью которого каждый поток будет обрабатывать собственную, независимую, часть матрицы.

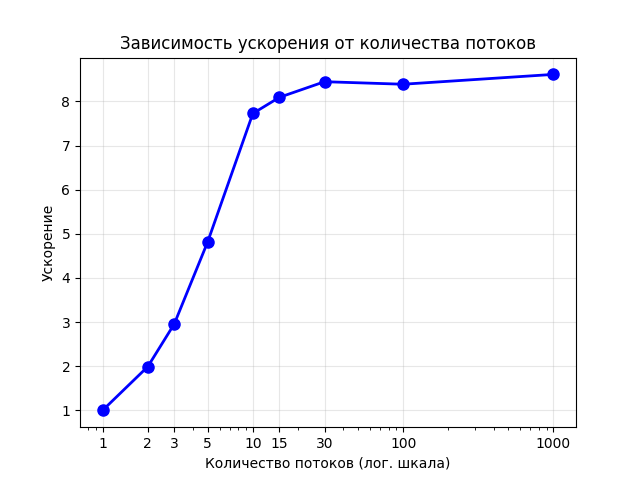
Так же в рамках лабораторной работы были собраны и проанализированы метрики зависимости времени, ускорения и эффективности от количества потоков.

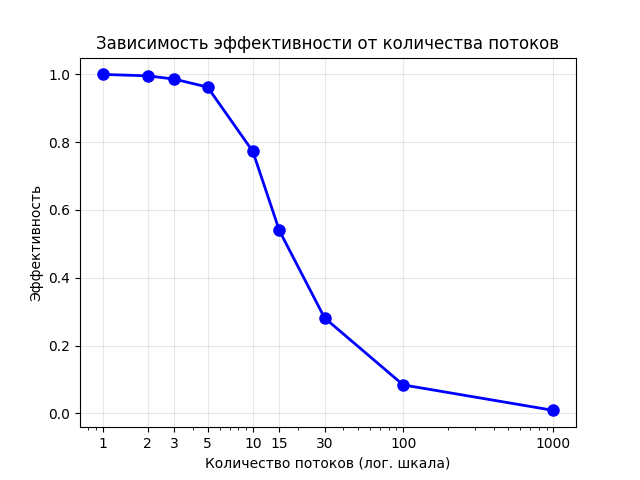
Ниже приведена табличка с результатами тестов наложения медианного фильтра 3х3 100 раз на матрице 1000х1000, состоящей из псевдорандомных чисел от 1 до 10000:

| Число потоков | Время исполнения (сек.) | Ускорение | Эффективность |
| --- | --- | --- | --- |
| 1 | 18.481 | 1.000 | 1.000 |
| 2 | 9.280 | 1.991 | 0.995 |
| 3 | 6.247 | 2.958 | 0.986 |
| 5 | 3.840 | 4.812 | 0.962 |
| 10 | 2.392 | 7.726 | 0.772 |
| 15 | 2.284 | 8.091 | 0.539 |
| 30 | 2.188 | 8.446 | 0.281 |
| 100 | 2.203 | 8.389 | 0.083 |
| 1000 | 2.146 | 8.661 | 0.008 |

Ниже приведены графики измерений из таблицы:







Формулы используемые в вычислениях:

Ускорение S:

,

где

Эффективность X:

где

**Код программы**

**generate\_test.c**

**#include <stdio.h>**

**#include <time.h>**

**#include <stdlib.h>**

**#include "generate\_test.h"**

**int64\_t get\_random\_number(int64\_t start, int64\_t end) {**

**return rand() % (end - start + 1) + start;**

**}**

**int64\_t\*\* generate\_random\_data(int64\_t rows\_matrix, int64\_t cols\_matrix, int64\_t min\_value, int64\_t max\_value) {**

**srand(time(NULL));**

**int64\_t\*\* data = (int64\_t\*\*)malloc(sizeof(int64\_t\*) \* rows\_matrix);**

**if (!data) {**

**printf("Failed to allocate memory for data\n");**

**return NULL;**

**}**

**for (int64\_t i = 0; i < rows\_matrix; ++i) {**

**data[i] = (int64\_t\*)malloc(sizeof(int64\_t) \* cols\_matrix);**

**if (!data[i]) {**

**for (int64\_t j = 0; j < i; ++j) {**

**free(data[j]);**

**}**

**free(data);**

**printf("Failed to allocate memory for data\n");**

**return NULL;**

**}**

**for (int64\_t j = 0; j < cols\_matrix; ++j) {**

**data[i][j] = get\_random\_number(min\_value, max\_value);**

**}**

**}**

**return data;**

**}**

**int64\_t\*\* get\_base\_data(int64\_t\* rows\_matrix, int64\_t\* cols\_matrix) {**

**int64\_t tmp[5][5] = {**

**{1, 10, 18, 44, 32},**

**{5, 104, 42, 11, 18},**

**{1, 9, 23, 19, 52},**

**{6, 10, 72, 12, 62},**

**{101, 76, 48, 24, 56}**

**};**

**\*rows\_matrix = 5;**

**\*cols\_matrix = 5;**

**int64\_t\*\* data = (int64\_t\*\*)malloc(\*rows\_matrix \* sizeof(int64\_t\*));**

**if (!data) {**

**printf("Failed to allocate memory for data\n");**

**return NULL;**

**}**

**for (int64\_t i = 0; i < \*rows\_matrix; ++i) {**

**data[i] = (int64\_t\*)malloc(\*cols\_matrix \* sizeof(int64\_t));**

**if (!data[i]) {**

**printf("Failed to allocate memory for data\n");**

**for (int64\_t j = 0; j < i; ++j) {**

**free(data[j]);**

**}**

**free(data);**

**return NULL;**

**}**

**for (int64\_t j = 0; j < \*cols\_matrix; ++j) {**

**data[i][j] = tmp[i][j];**

**}**

**}**

**return data;**

**}**

**main.c**

**#include <stdio.h>**

**#include <stdlib.h>**

**#include <unistd.h>**

**#include <sys/time.h>**

**#include <pthread.h>**

**#include <string.h>**

**#include "generate\_test.h"**

**typedef struct {**

**int64\_t\*\* matrix;**

**int64\_t\*\* result;**

**int64\_t id;**

**int64\_t n\_threads;**

**int64\_t iterations;**

**int64\_t rows;**

**int64\_t cols;**

**int64\_t window\_rows;**

**int64\_t window\_cols;**

**} ThreadArgs;**

**int cmp(const void\* a, const void\* b) {**

**int64\_t ia = \*(int64\_t\*)a;**

**int64\_t ib = \*(int64\_t\*)b;**

**if (ia < ib)**

**return -1;**

**if (ia > ib)**

**return 1;**

**return 0;**

**}**

**void app\_median\_filter(int64\_t\*\* matrix, int64\_t\*\* result, int64\_t rows, int64\_t cols, int64\_t start\_rows, int64\_t end\_rows,**

**int64\_t start\_cols,**

**int64\_t end\_cols, int64\_t window\_rows, int64\_t window\_cols) {**

**int64\_t sz = window\_rows \* window\_cols;**

**int64\_t\* tmp = (int64\_t\*)malloc(sz \* sizeof(int64\_t));**

**if (!tmp) {**

**printf("malloc failed\n");**

**return;**

**}**

**int64\_t shift\_rows = window\_rows / 2;**

**int64\_t shift\_cols = window\_cols / 2;**

**int64\_t median;**

**for (int64\_t i = start\_rows; i < end\_rows; ++i) {**

**for (int64\_t j = start\_cols; j < end\_cols; ++j) {**

**if (i - shift\_rows < 0 || i + shift\_rows >= rows || j - shift\_cols < 0 || j + shift\_cols >= cols) {**

**// клетки, которые не могут быть центральным значением в окне просто переписываем из исходной матрицы**

**result[i][j] = matrix[i][j];**

**continue;**

**}**

**int64\_t index = 0;**

**int64\_t offset\_rows = (window\_rows % 2 == 0) ? 1 : 0;**

**int64\_t offset\_cols = (window\_cols % 2 == 0) ? 1 : 0;**

**for (int64\_t k = -shift\_rows; k <= shift\_rows - offset\_rows; ++k) {**

**for (int64\_t l = -shift\_cols; l <= shift\_cols - offset\_cols; ++l) {**

**tmp[index++] = matrix[i + k][j + l];**

**}**

**}**

**qsort(tmp, sz, sizeof(int64\_t), cmp);**

**if (sz % 2 == 1)**

**median = tmp[sz / 2];**

**else**

**median = (tmp[sz / 2 - 1] + tmp[sz / 2]) / 2; // псевдомедиана**

**result[i][j] = median;**

**}**

**}**

**free(tmp);**

**}**

**static void\* calculate(void\* \_args) {**

**ThreadArgs\* args = (ThreadArgs\*)\_args;**

**int64\_t chunk\_rows = args->rows / args->n\_threads;**

**int64\_t start\_rows = args->id \* chunk\_rows;**

**int64\_t end\_rows = (args->id == args->n\_threads - 1) ? args->rows : start\_rows + chunk\_rows;**

**for (int64\_t i = 0; i < args->iterations; ++i) {**

**app\_median\_filter(args->matrix, args->result, args->rows, args->cols, start\_rows, end\_rows, 0, args->cols,**

**args->window\_rows,**

**args->window\_cols);**

**}**

**return NULL;**

**}**

**double get\_time() {**

**struct timeval tv;**

**gettimeofday(&tv, NULL);**

**return tv.tv\_sec + tv.tv\_usec / 1000000.0;**

**}**

**int main(int argc, char\*\* argv) {**

**if (argc != 4) {**

**printf("Usage: %s <flag> <target\_file> <n\_threads>\n", argv[0]);**

**return 1;**

**}**

**const char\* flag = argv[1];**

**const char\* filename = argv[2];**

**int64\_t n\_threads = strtol(argv[3], NULL, 10);**

**if (n\_threads < 1) {**

**printf("Invalid number of arguments\n");**

**return 1;**

**}**

**int64\_t rows\_matrix, cols\_matrix, min\_value, max\_value;**

**int64\_t\*\* matrix;**

**if (strcmp("-r", flag) == 0) {**

**printf("<rows\_matrix>: ");**

**scanf("%lld", &rows\_matrix);**

**printf("<cols\_matrix>: ");**

**scanf("%lld", &cols\_matrix);**

**printf("<min\_value>: ");**

**scanf("%lld", &min\_value);**

**printf("<max\_value>: ");**

**scanf("%lld", &max\_value);**

**matrix = generate\_random\_data(rows\_matrix, cols\_matrix, min\_value, max\_value);**

**} else if (strcmp("-b", flag) == 0) {**

**matrix = get\_base\_data(&rows\_matrix, &cols\_matrix);**

**} else {**

**printf("Invalid flag\n");**

**return 1;**

**}**

**int64\_t iters, length, width;**

**printf("<number of repetitions>: ");**

**scanf("%lld", &iters);**

**printf("<median filter window length>: ");**

**scanf("%lld", &length);**

**printf("<median filter window width>: ");**

**scanf("%lld", &width);**

**if (rows\_matrix < 0 || cols\_matrix < 0) {**

**printf("matrix sides must be positive\n");**

**return 1;**

**}**

**if (length > rows\_matrix || width > cols\_matrix) {**

**printf("Size the window must be less then matrix");**

**return 1;**

**}**

**if (!matrix) {**

**printf("Failed to generate data\n");**

**return 1;**

**}**

**int64\_t\*\* result = (int64\_t\*\*)malloc(sizeof(int64\_t\*) \* rows\_matrix);**

**if (!result) {**

**printf("malloc failed\n");**

**return 1;**

**}**

**for (int64\_t i = 0; i < rows\_matrix; ++i) {**

**result[i] = (int64\_t\*)malloc(sizeof(int64\_t) \* cols\_matrix);**

**if (!result[i]) {**

**printf("malloc failed\n");**

**for (int64\_t j = 0; j < i; ++j) {**

**free(result[j]);**

**}**

**free(result);**

**return 1;**

**}**

**}**

**FILE\* f = fopen(filename, "w");**

**if (!f) {**

**printf("Failed to open file\n");**

**for (int64\_t i = 0; i < rows\_matrix; ++i) {**

**free(result[i]);**

**free(matrix[i]);**

**}**

**free(result);**

**free(matrix);**

**return 1;**

**}**

**fprintf(f, "First matrix:\n");**

**for (int64\_t i = 0; i < rows\_matrix; ++i) {**

**for (int64\_t j = 0; j < cols\_matrix; ++j) {**

**fprintf(f, "%lld ", matrix[i][j]);**

**}**

**fprintf(f, "\n");**

**}**

**pthread\_t\* threads = (pthread\_t\*)malloc(n\_threads \* sizeof(pthread\_t));**

**ThreadArgs\* thread\_args = (ThreadArgs\*)malloc(n\_threads \* sizeof(ThreadArgs));**

**if (!threads || !thread\_args) {**

**printf("Failed to allocate memory for threads\n");**

**for (int64\_t i = 0; i < rows\_matrix; ++i) {**

**free(matrix[i]);**

**free(result[i]);**

**}**

**free(thread\_args);**

**free(matrix);**

**free(result);**

**return 1;**

**}**

**double start = get\_time();**

**for (int64\_t i = 0; i < n\_threads; ++i) {**

**thread\_args[i] = (ThreadArgs){**

**.matrix = matrix,**

**.result = result,**

**.id = i,**

**.n\_threads = n\_threads,**

**.iterations = iters,**

**.rows = rows\_matrix,**

**.cols = cols\_matrix,**

**.window\_rows = length,**

**.window\_cols = width**

**};**

**pthread\_create(&threads[i], NULL, calculate, &thread\_args[i]);**

**}**

**for (int64\_t i = 0; i < n\_threads; ++i) {**

**pthread\_join(threads[i], NULL);**

**}**

**double end = get\_time();**

**fprintf(f, "Second matrix:\n");**

**for (int64\_t i = 0; i < rows\_matrix; ++i) {**

**for (int64\_t j = 0; j < cols\_matrix; ++j) {**

**fprintf(f, "%lld ", result[i][j]);**

**}**

**fprintf(f, "\n");**

**}**

**printf("Total running time on %ld threads: %.3lf \n", n\_threads, end - start);**

**for (int64\_t i = 0; i < rows\_matrix; ++i) {**

**free(matrix[i]);**

**free(result[i]);**

**}**

**free(thread\_args);**

**free(threads);**

**free(result);**

**free(matrix);**

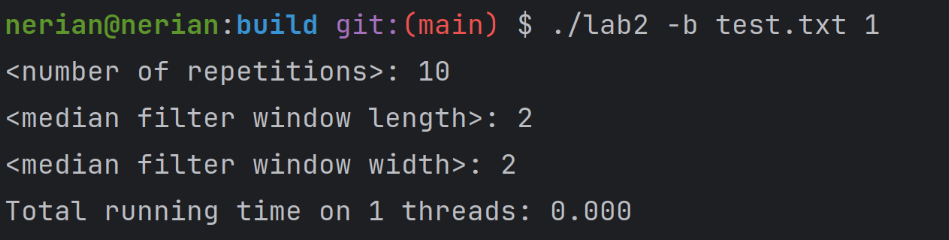
**fclose(f);**

**return 0;**

**}**

**Протокол работы программы**

*TEST1*

****

**First matrix:**

**1 10 18 44 32**

**5 104 42 11 18**

**1 9 23 19 52**

**6 10 72 12 62**

**101 76 48 24 56**

**Second matrix:**

**1 10 18 44 32**

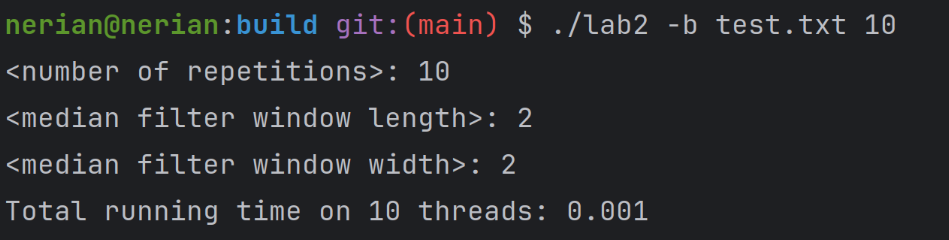
**5 7 30 30 18**

**1 7 32 21 52**

**6 7 16 21 62**

**101 76 48 24 56**

*TEST2*

****

**First matrix:**

**1 10 18 44 32**

**5 104 42 11 18**

**1 9 23 19 52**

**6 10 72 12 62**

**101 76 48 24 56**

**Second matrix:**

**1 10 18 44 32**

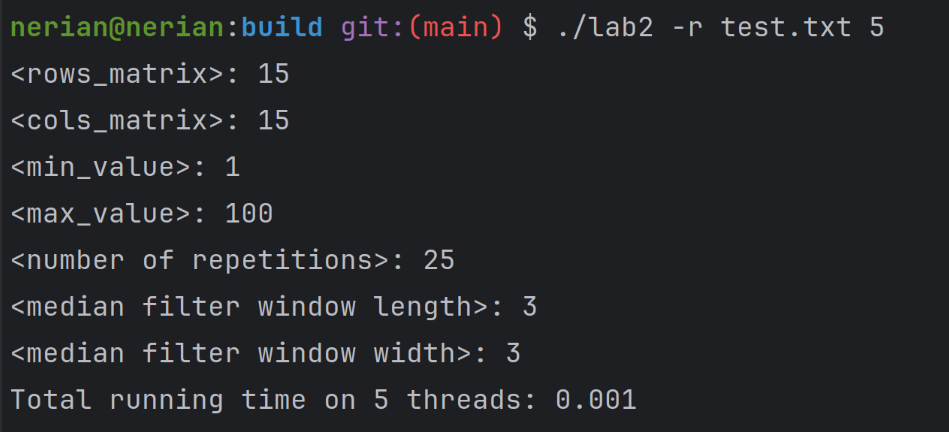
**5 7 30 30 18**

**1 7 32 21 52**

**6 7 16 21 62**

**101 76 48 24 56**

*TEST3*

**

First matrix:

25 24 87 45 7 79 40 79 1 66 82 84 70 18 46

34 61 25 13 62 3 68 60 66 99 52 44 25 89 37

59 65 12 45 10 19 75 49 97 75 66 31 10 87 48

55 20 60 32 85 74 34 4 85 51 2 36 94 79 25

82 37 41 93 33 50 63 7 50 60 33 67 42 42 53

41 48 25 100 79 9 73 12 64 57 14 66 45 59 96

21 40 32 61 85 16 63 47 74 12 58 6 79 51 99

83 91 46 59 43 25 19 15 88 83 24 54 100 20 12

95 40 4 78 100 88 93 62 86 66 26 44 23 56 94

21 38 85 18 49 27 94 67 93 82 49 68 87 48 87

98 94 26 53 71 78 40 15 91 26 32 16 21 54 71

66 26 61 2 44 9 80 37 27 25 70 76 92 56 75

31 6 21 8 58 91 37 50 58 28 27 89 43 99 95

14 64 20 26 66 15 34 45 4 60 69 73 87 13 81

14 95 38 34 54 95 76 91 96 33 18 22 22 12 20

Second matrix:

25 24 87 45 7 79 40 79 1 66 82 84 70 18 46

34 34 45 25 19 40 60 66 66 66 66 52 44 46 37

59 55 32 32 32 62 49 66 66 66 51 36 44 48 48

55 55 41 41 45 50 49 50 51 60 51 36 42 48 25

82 41 41 60 74 63 34 50 51 51 51 42 59 53 53

41 40 41 61 61 63 47 63 50 57 57 45 51 53 96

21 41 48 59 59 43 19 63 57 58 54 54 54 59 99

83 40 46 59 61 63 47 63 66 66 44 44 51 56 12

95 46 46 49 49 49 62 86 82 82 54 49 54 56 94

21 40 40 53 71 78 67 86 67 66 44 32 48 56 87

98 61 38 49 44 49 40 67 37 49 49 68 56 71 71

66 31 26 44 53 58 40 40 28 28 28 43 56 71 75

31 26 21 26 26 44 37 37 37 28 69 73 76 81 95

14 21 26 34 54 58 50 50 50 33 33 43 43 43 81

14 95 38 34 54 95 76 91 96 33 18 22 22 12 20

**Вывод**

При выполнении лабораторной работы я научился работать с потоками и ускорять программы последовательного выполнения при помощи многопоточности. Также изучил различные метрики для оценки программы. Наибольшие трудности были в реализации вычислений (fork) для каждого потока и получении корректного результата (join) при различных входных данных.