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-206Б-23

Студент: Абдыкалыков Н. А.

Преподаватель: Миронов Е. С.

Оценка: _____

Дата: 02.03.25

Москва, 2025

Постановка задачи

Вариант 13.

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.

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

Общий метод и алгоритм решения

Использованные системные вызовы и функции:

pthread_create – создает новый поток для выполнения задачи.

pthread_join – ожидает завершения работы потока.

clock_gettime – используется для замера времени выполнения программы.

malloc – выделяет память для матриц и временных данных.

free – освобождает выделенную память.

rand – генерирует случайные числа для заполнения матриц.

Алгоритм решения:

Инициализация:

Программа получает от пользователя размеры матрицы (rows, cols), размер ядра свёртки (kernelSize), количество итераций (K) и максимальное количество потоков (maxThreads). Далее Исходная матрица и ядро свертки заполняются случайными вещественными числами.

Многопоточная обработка:

Задача разбивается на несколько потоков. Каждый поток обрабатывает свою часть строк исходной матрицы. Потоки создаются с помощью pthread_create.

Операция свертки:

Каждый поток выполняет свертку для своей части матрицы, используя заданное ядро. Для каждого элемента результирующей матрицы вычисляется взвешенная сумма элементов исходной матрицы, покрываемых ядром свертки. Если ядро выходит за границы матрицы, такие элементы игнорируются.

Повторение K раз:

Операция свертки применяется K раз. На каждой итерации результат предыдущей свертки становится входной матрицей для следующей итерации

Тестирование

Программа была протестирована на входных данных: матрица 200 на 200 из случайных элементов, 5 итераций.

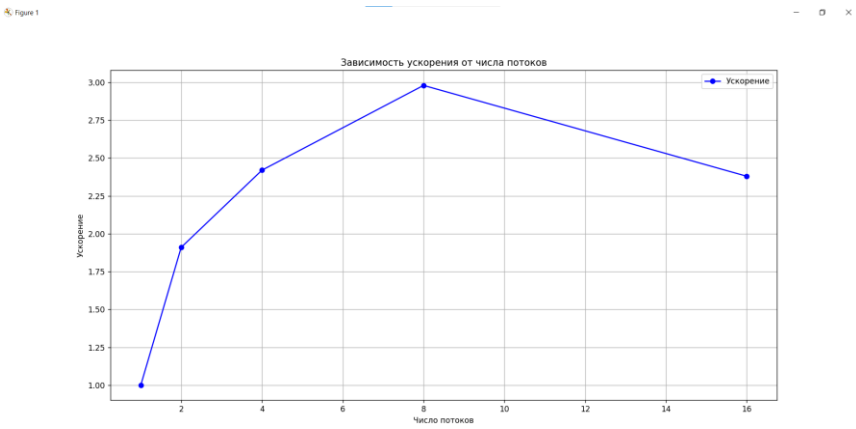
Формулы:

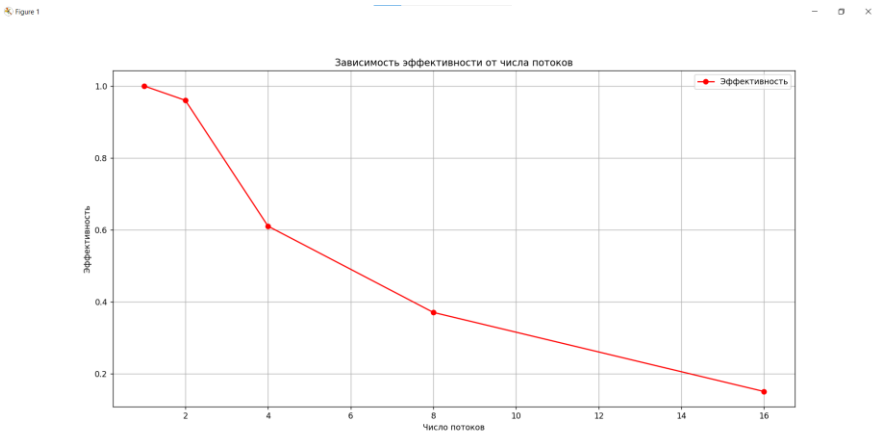
Ускорение = Время выполнения на N потоках / Время выполнения на 1 потоке

Эффективность = Количество потоков / Ускорение

Число потоков	Время исполнения (мс)	Ускорение	Эффективность
1	0.011182	1.00	1.00
2	0.005844	1.91	0.96
4	0.004623	2.42	0.61
8	0.003750	2.98	0.37
16	0.004707	2.38	0.15

Графики:





Анализ результатов:

Результаты показывают, что увеличение числа потоков сначала приводит к значительному ускорению выполнения программы. Наибольшее ускорение достигается при **8 потоках** (ускорение в **2.98 раза** по сравнению с одним потоком). Однако при дальнейшем

увеличении числа потоков (до 16) ускорение снижается до **2.38**, а эффективность падает до **0.15**.

Это связано с **накладными расходами** на управление потоками и **конкуренцией за ресурсы** (например, доступ к памяти или кэшу процессора). При большом количестве потоков накладные расходы начинают преобладать над выигрышем от параллелизма.

Оптимальное распределение достигается при **4-8 потоках**, где ускорение максимально, а эффективность остается на приемлемом уровне (0.61 для 4 потоков и 0.37 для 8 потоков). Дальнейшее увеличение числа потоков (например, до 16) не приводит к значительному приросту производительности и снижает эффективность использования ресурсов.

Код программы:

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <stdint.h>
#include <time.h>

typedef struct {
    double* input;
    double* output;
    double* kernel;
    int rows;
    int cols;
    int kernelSize;
    int rowStart;
    int rowEnd;
} ThreadData;

void* threadConvolution(void* lpParam)
{
    ThreadData* td = (ThreadData*)lpParam;

    int kHalf = td->kernelSize / 2;
    for (int r = td->rowStart; r < td->rowEnd; r++) {
        for (int c = 0; c < td->cols; c++) {
            double sum = 0.0;
```

```
            for (int kr = 0; kr < td->kernelSize; kr++) {
                for (int kc = 0; kc < td->kernelSize; kc++) {
                    int rr = r + kr - kHalf;
                    int cc = c + kc - kHalf;

                    if (rr >= 0 && rr < td->rows && cc >= 0 && cc < td->cols) {
                        sum += td->input[rr * td->cols + cc]
                                * td->kernel[kr * td->kernelSize + kc];
                    }
                }
            }
            td->output[r * td->cols + c] = sum;
        }
    }
    return NULL;
}

void applyConvolution2D(
    double* input,
    double* output,
    double* kernel,
    int rows,
    int cols,
    int kernelSize,
    int maxThreads)
{
    int numThreads = (rows < maxThreads) ? rows : maxThreads;

    int chunkSize = rows / numThreads;
    int remainder = rows % numThreads;

    pthread_t* threads = (pthread_t*)malloc(numThreads * sizeof(pthread_t));
    ThreadData* td = (ThreadData*)malloc(numThreads * sizeof(ThreadData));
```

```
    int currentStart = 0;

    for (int i = 0; i < numThreads; i++) {
        int start = currentStart;
        int size = chunkSize + ((i < remainder) ? 1 : 0);
        int end = start + size;

        td[i].input = input;
        td[i].output = output;
        td[i].kernel = kernel;
        td[i].rows = rows;
        td[i].cols = cols;
        td[i].kernelSize = kernelSize;
        td[i].rowStart = start;
        td[i].rowEnd = end;

        pthread_create(&threads[i], NULL, threadConvolution, &td[i]);

        currentStart = end;
    }

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

    free(threads);
    free(td);
}

int main(int argc, char* argv[])
{
    if (argc < 6) {
        printf("Usage: %s <rows> <cols> <kernelSize> <K> <maxThreads>\n", argv[0]);
        return 1;
```

```
}

int rows = atoi(argv[1]);
int cols = atoi(argv[2]);
int kernelSize = atoi(argv[3]);
int K = atoi(argv[4]);
int maxThreads = atoi(argv[5]);

if (rows <= 0 || cols <= 0 || kernelSize <= 0 || K <= 0 || maxThreads <= 0) {
    printf("Error: invalid arguments\n");
    return 1;
}

double* matrix = (double*)malloc(rows * cols * sizeof(double));
double* temp   = (double*)malloc(rows * cols * sizeof(double));

 // Инициализация генератора случайных чисел
srand(12345);
for (int r = 0; r < rows; r++) {
    for (int c = 0; c < cols; c++) {
        matrix[r * cols + c] = (double)(rand() % 100) / 10.0;
    }
}

printf("Matrix size: %dx%d\n", rows, cols);
printf("Kernel size: %d\n", kernelSize);
printf("Number of iterations (K): %d\n", K);
printf("Maximum number of threads: %d\n", maxThreads);
printf("\n");

// Вывод начальной матрицы
printf("Initial matrix:\n");
for (int r = 0; r < rows; r++) {
    for (int c = 0; c < cols; c++) {
```

```
            printf("%6.2f ", matrix[r * cols + c]);
        }
        printf("\n");
    }
    printf("\n");

    double* kernel = (double*)malloc(kernelSize * kernelSize * sizeof(double));
    for (int i = 0; i < kernelSize * kernelSize; i++) {
        kernel[i] = (double)(rand() % 10) / 10.0;
    }

    // Замер времени начала выполнения
    struct timespec start, end;
    clock_gettime(CLOCK_MONOTONIC, &start);

    for (int i = 0; i < K; i++) {
        applyConvolution2D(matrix, temp, kernel, rows, cols, kernelSize, maxThreads);
        double* swap = matrix;
        matrix = temp;
        temp = swap;
    }

    // Замер времени окончания выполнения
    clock_gettime(CLOCK_MONOTONIC, &end);

    // Вычисление затраченного времени в секундах
    double time_taken = (end.tv_sec - start.tv_sec) + (end.tv_nsec - start.tv_nsec) / 1e9;

    printf("Resulting matrix:\n");
    for (int r = 0; r < rows; r++) {
        for (int c = 0; c < cols; c++) {
            printf("%6.2f ", matrix[r * cols + c]);
        }
        printf("\n");
```

```
    }

    printf("\nTime taken: %.6f seconds\n", time_taken);

    free(matrix);
    free(temp);
    free(kernel);

    return 0;
}
```

Протокол работы программы

$ ./a.out 10 10 3 1 1

Matrix size: 10x10

Kernel size: 3

Number of iterations (K): 1

Maximum number of threads: 1

Initial matrix:

9.90	2.10	7.30	5.10	2.10	0.40	3.70	0.20	2.80	2.80
2.70	9.70	2.10	5.30	0.60	7.30	3.80	1.20	5.70	1.20
6.10	9.50	0.40	4.20	2.30	7.90	2.70	8.60	6.50	4.30
7.70	6.40	1.70	5.10	6.70	3.80	5.50	0.40	4.10	8.30
3.20	6.80	3.20	5.30	2.10	3.80	7.80	1.10	5.00	3.50
7.50	6.30	3.00	7.90	0.50	5.30	1.10	3.20	9.10	7.60
2.80	2.10	9.20	9.70	7.20	5.90	3.50	7.90	6.30	7.60
1.40	4.70	9.60	4.60	5.20	6.90	8.40	3.00	3.20	3.40
6.50	0.80	9.80	4.80	3.90	5.50	5.30	5.00	4.00	4.50
7.80	2.00	6.60	2.20	1.70	9.00	8.10	0.40	6.90	9.60

Resulting matrix:

10.60	12.90	10.24	5.99	6.94	9.21	4.02	7.32	6.07	1.73
20.78	15.60	14.35	9.68	15.16	12.12	11.44	16.18	10.95	5.17
21.25	11.77	15.18	12.00	18.12	13.60	14.42	13.65	15.09	7.71

18.19 13.44 15.44 12.34 13.98 16.88 13.74 14.93 18.00 6.73

18.75 14.32 17.35 12.31 13.59 15.42 8.24 16.29 18.75 9.28

12.39 15.85 23.71 15.85 17.37 13.14 14.78 20.68 19.39 8.65

11.14 21.78 23.15 17.35 18.88 15.90 16.98 15.97 18.51 8.42

9.13 21.31 21.23 20.44 20.11 19.62 17.10 15.78 16.71 7.48

9.28 20.62 16.05 14.00 18.48 22.82 15.90 14.69 17.82 8.32

4.53 12.69 7.98 8.40 12.05 12.21 7.73 10.77 12.40 4.20

Time taken: 0.000105 seconds

$ ./a.out 10 10 3 1 4

Matrix size: 10x10

Kernel size: 3

Number of iterations (K): 1

Maximum number of threads: 4

Initial matrix:

9.90 2.10 7.30 5.10 2.10 0.40 3.70 0.20 2.80 2.80

2.70 9.70 2.10 5.30 0.60 7.30 3.80 1.20 5.70 1.20

6.10 9.50 0.40 4.20 2.30 7.90 2.70 8.60 6.50 4.30

7.70 6.40 1.70 5.10 6.70 3.80 5.50 0.40 4.10 8.30

3.20 6.80 3.20 5.30 2.10 3.80 7.80 1.10 5.00 3.50

7.50 6.30 3.00 7.90 0.50 5.30 1.10 3.20 9.10 7.60

2.80 2.10 9.20 9.70 7.20 5.90 3.50 7.90 6.30 7.60

1.40 4.70 9.60 4.60 5.20 6.90 8.40 3.00 3.20 3.40

6.50 0.80 9.80 4.80 3.90 5.50 5.30 5.00 4.00 4.50

7.80 2.00 6.60 2.20 1.70 9.00 8.10 0.40 6.90 9.60

Resulting matrix:

10.60 12.90 10.24 5.99 6.94 9.21 4.02 7.32 6.07 1.73

20.78 15.60 14.35 9.68 15.16 12.12 11.44 16.18 10.95 5.17

21.25 11.77 15.18 12.00 18.12 13.60 14.42 13.65 15.09 7.71

18.19 13.44 15.44 12.34 13.98 16.88 13.74 14.93 18.00 6.73

18.75 14.32 17.35 12.31 13.59 15.42 8.24 16.29 18.75 9.28

12.39 15.85 23.71 15.85 17.37 13.14 14.78 20.68 19.39 8.65

11.14 21.78 23.15 17.35 18.88 15.90 16.98 15.97 18.51 8.42

9.13 21.31 21.23 20.44 20.11 19.62 17.10 15.78 16.71 7.48

9.28 20.62 16.05 14.00 18.48 22.82 15.90 14.69 17.82 8.32

4.53 12.69 7.98 8.40 12.05 12.21 7.73 10.77 12.40 4.20

Time taken: 0.000252 seconds

Strace:

execve("./a.out", ["./a.out", "10", "10", "3", "1", "4"], 0x7ffee38bfdd8 /* 22 vars */) = 0

brk(NULL) = 0x557de9c7c000

mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f4e681d6000

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

newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=18354, ...}, AT_EMPTY_PATH) = 0

mmap(NULL, 18354, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f4e681d1000

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\20t\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

newfstatat(3, "", {st_mode=S_IFREG|0755, st_size=1922136, ...}, 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, 1970000, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f4e67ff0000

mmap(0x7f4e68016000, 1396736, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x26000) = 0x7f4e68016000

mmap(0x7f4e6816b000, 339968, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x17b000) = 0x7f4e6816b000

mmap(0x7f4e681be000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1ce000) = 0x7f4e681be000

mmap(0x7f4e681c4000, 53072, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7f4e681c4000

close(3) = 0

mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f4e67fed000

arch_prctl(ARCH_SET_FS, 0x7f4e67fed740) = 0

set_tid_address(0x7f4e67feda10) = 3433

```
set_robust_list(0x7f4e67feda20, 24)     = 0
rseq(0x7f4e67fee060, 0x20, 0, 0x53053053) = 0
mprotect(0x7f4e681be000, 16384, PROT_READ) = 0
mprotect(0x557dd36cb000, 4096, PROT_READ) = 0
mprotect(0x7f4e68208000, 8192, PROT_READ) = 0
prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0
munmap(0x7f4e681d1000, 18354)           = 0
getrandom("\x0e\x16\xac\xd0\x88\x99\xa0\x81", 8, GRND_NONBLOCK) = 8
brk(NULL)                               = 0x557de9c7c000
brk(0x557de9c9d000)                     = 0x557de9c9d000
newfstatat(1, "", {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0), ...}, AT_EMPTY_PATH) = 0
write(1, "Matrix size: 10x10\n", 19Matrix size: 10x10
)    = 19
write(1, "Kernel size: 3\n", 15Kernel size: 3
)       = 15
write(1, "Number of iterations (K): 1\n", 28Number of iterations (K): 1
) = 28
write(1, "Maximum number of threads: 4\n", 29Maximum number of threads: 4
) = 29
write(1, "\n", 1
)                       = 1
write(1, "Initial matrix:\n", 16Initial matrix:
)      = 16
write(1, " 9.90  2.10  7.30  5.10  2."..., 71 9.90  2.10  7.30  5.10  2.10  0.40  3.70  0.20  2.80
2.80
) = 71
write(1, " 2.70  9.70  2.10  5.30  0."..., 71 2.70  9.70  2.10  5.30  0.60  7.30  3.80  1.20  5.70
1.20
) = 71
write(1, " 6.10  9.50  0.40  4.20  2."..., 71 6.10  9.50  0.40  4.20  2.30  7.90  2.70  8.60  6.50
4.30
) = 71
write(1, " 7.70  6.40  1.70  5.10  6."..., 71 7.70  6.40  1.70  5.10  6.70  3.80  5.50  0.40  4.10
8.30
) = 71
```

write(1, " 3.20 6.80 3.20 5.30 2."..., 71 3.20 6.80 3.20 5.30 2.10 3.80 7.80 1.10 5.00 3.50

) = 71

write(1, " 7.50 6.30 3.00 7.90 0."..., 71 7.50 6.30 3.00 7.90 0.50 5.30 1.10 3.20 9.10 7.60

) = 71

write(1, " 2.80 2.10 9.20 9.70 7."..., 71 2.80 2.10 9.20 9.70 7.20 5.90 3.50 7.90 6.30 7.60

) = 71

write(1, " 1.40 4.70 9.60 4.60 5."..., 71 1.40 4.70 9.60 4.60 5.20 6.90 8.40 3.00 3.20 3.40

) = 71

write(1, " 6.50 0.80 9.80 4.80 3."..., 71 6.50 0.80 9.80 4.80 3.90 5.50 5.30 5.00 4.00 4.50

) = 71

write(1, " 7.80 2.00 6.60 2.20 1."..., 71 7.80 2.00 6.60 2.20 1.70 9.00 8.10 0.40 6.90 9.60

) = 71

write(1, "\n", 1

) = 1

rt_sigaction(SIGRT_1, {sa_handler=0x7f4e68076720, sa_mask=[], sa_flags=SA_RESTORER|SA_ONSTACK|SA_RESTART|SA_SIGINFO, sa_restorer=0x7f4e6802c050}, 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) = 0x7f4e677ec000

mprotect(0x7f4e677ed000, 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=0x7f4e67fec990, parent_tid=0x7f4e67fec990, exit_signal=0, stack=0x7f4e677ec000, stack_size=0x7fff80, tls=0x7f4e67fec6c0} => {parent_tid=[3434]}, 88) = 3434

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

mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) = 0x7f4e66feb000

mprotect(0x7f4e66fec000, 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=0x7f4e677eb990, parent_tid=0x7f4e677eb990, exit_signal=0, stack=0x7f4e66feb000, stack_size=0x7fff80, tls=0x7f4e677eb6c0} => {parent_tid=[3435]}, 88) = 3435

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

mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) = 0x7f4e667ea000

mprotect(0x7f4e667eb000, 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=0x7f4e66fea990, parent_tid=0x7f4e66fea990, exit_signal=0, stack=0x7f4e667ea000, stack_size=0x7fff80, tls=0x7f4e66fea6c0} => {parent_tid=[0]}, 88) = 3436

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

mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) = 0x7f4e65fe9000

mprotect(0x7f4e65fea000, 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=0x7f4e667e9990, parent_tid=0x7f4e667e9990, exit_signal=0, stack=0x7f4e65fe9000, stack_size=0x7fff80, tls=0x7f4e667e96c0} => {parent_tid=[0]}, 88) = 3437

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

write(1, "Resulting matrix:\n", 18Resulting matrix:

)   = 18

write(1, " 10.60 12.90 10.24   5.99   6."..., 71 10.60 12.90 10.24   5.99   6.94   9.21   4.02   7.32   6.07   1.73

) = 71

write(1, " 20.78 15.60 14.35   9.68 15."..., 71 20.78 15.60 14.35   9.68 15.16 12.12 11.44 16.18 10.95   5.17

) = 71

write(1, " 21.25 11.77 15.18 12.00 18."..., 71 21.25 11.77 15.18 12.00 18.12 13.60 14.42 13.65 15.09   7.71

) = 71

write(1, " 18.19 13.44 15.44 12.34 13."..., 71 18.19 13.44 15.44 12.34 13.98 16.88 13.74 14.93 18.00   6.73

) = 71

write(1, " 18.75 14.32 17.35 12.31 13."..., 71 18.75 14.32 17.35 12.31 13.59 15.42   8.24 16.29 18.75   9.28

) = 71

write(1, " 12.39 15.85 23.71 15.85 17."..., 71 12.39 15.85 23.71 15.85 17.37 13.14 14.78 20.68 19.39   8.65

) = 71

write(1, " 11.14 21.78 23.15 17.35 18."..., 71 11.14 21.78 23.15 17.35 18.88 15.90 16.98 15.97 18.51   8.42

) = 71

write(1, " 9.13 21.31 21.23 20.44 20."..., 71 9.13 21.31 21.23 20.44 20.11 19.62 17.10 15.78 16.71   7.48

) = 71

write(1, " 9.28 20.62 16.05 14.00 18."..., 71 9.28 20.62 16.05 14.00 18.48 22.82 15.90 14.69 17.82   8.32

) = 71

write(1, " 4.53 12.69   7.98   8.40 12."..., 71 4.53 12.69   7.98   8.40 12.05 12.21   7.73 10.77 12.40   4.20

) = 71

write(1, "\n", 1

)   = 1

write(1, "Time taken: 0.002646 seconds\n", 29Time taken: 0.002646 seconds

) = 29

exit_group(0)   = ?

+++ exited with 0 +++

Вывод

В ходе работы я освоил новые аспекты работы с языком **C**, включая создание и управление потоками с помощью библиотеки **pthread**, а также научился применять многопоточность для ускорения вычислений. Я углубил понимание того, как накладные расходы и конкуренция за ресурсы влияют на производительность программы, и научился анализировать результаты с помощью замеров времени и расчетов ускорения и эффективности. Что касается **Python**, то я вспомнил основы с первого курса, чтобы визуализировать результаты с помощью библиотеки **matplotlib**.