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: Стаценко В.А.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 24.12.24

Москва, 2024

Постановка задачи

Вариант 10.

Решить систему линейных уравнений методом Гаусса.

Общий метод и алгоритм решения

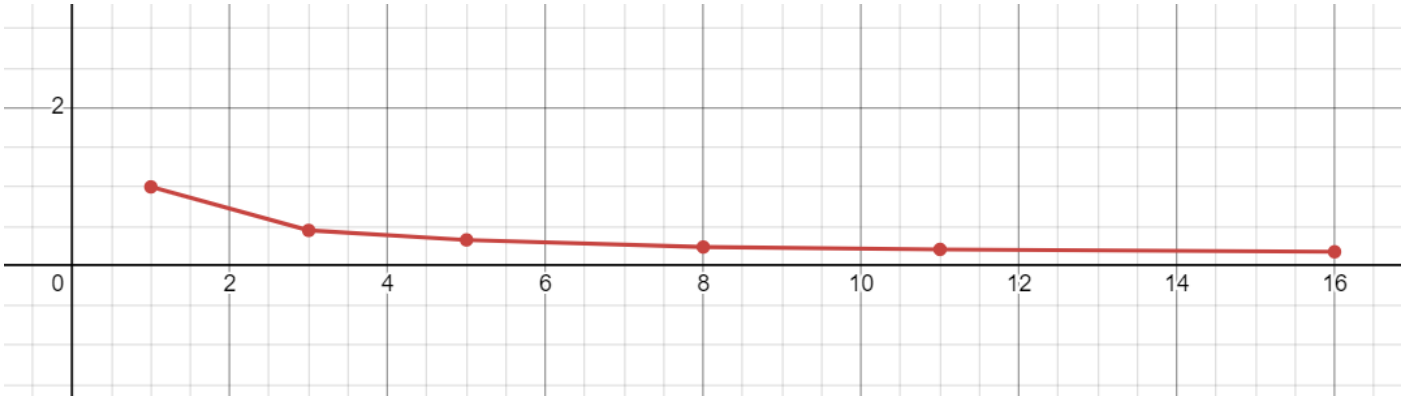
Использованные системные вызовы:

- int pthread_mutex_lock(pthread_mutex_t *mutex) - блокировка мьютекса;
- int pthread_mutex_unlock(pthread_mutex_t *mutex) - разблокировка мьютекса;
- int pthread_create(pthread_t *thread, const pthread_attr_t *attr, void*(*start_routine) (void *), void *arg) - создание потока;
- int pthread_mutex_init(pthread_mutex_t *mutex, const pthread_mutexattr_t *attr) - инициализирует мьютекс;
- int pthread_join(pthread_t thread, void ** retval) - ожидание завершения потока;
- void pthread_exit(void *retval) - завершает выполнение текущего потока;
- void exit(int status) - завершение программы с заданным кодом возврата.

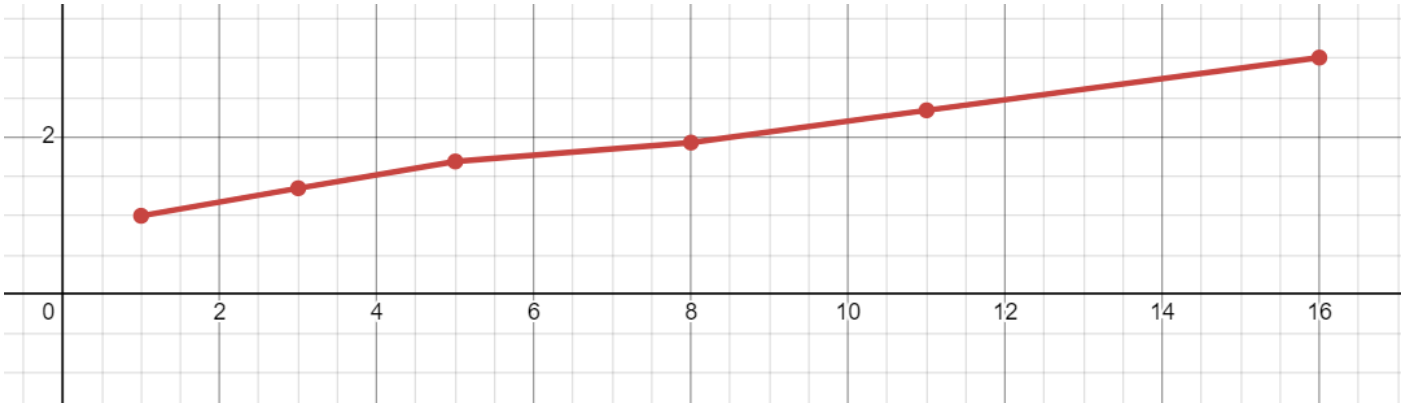
В данной лабораторной работе я написала программу для решения системы линейных уравнений Ax=b методом Гаусса, где сначала считываются размер матрицы n и максимальное количество потоков, затем выделяется память для матрицы A, вектора b и вектора решения x, которые заполняются случайными значениями; матрица A разделяется на части, каждая из которых обрабатывается отдельным потоком, для чего создаются структуры с данными о диапазоне строк для каждого потока, и потоки создаются; каждый поток выполняет прямой ход метода Гаусса для своей части матрицы, используя операции для управления количеством активных потоков, и если количество активных потоков превышает заданный лимит, поток повторяет итерацию; после завершения всех потоков основной поток ожидает их завершения, затем выполняется обратная подстановка для нахождения решения x, которое выводится на экран, и в конце освобождается выделенная память для матрицы A, вектора b и вектора решения x.

Число потоков	Время выполнения (мс)	Ускорение	Эффективность
1	942	1	1,00
3	697	1,35	0,45
5	555	1,69	0,33
8	486	1,93	0,24
11	401	2,34	0,21
16	312	3,01	0,18

Изменение эффективности:



Изменение ускорения:



Код программы

atomic.c:

```c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <stdatomic.h>

#define MAX_THREADS 16

typedef struct {
    int n;
    double **A;
    double *b;
    int start, end;
} ThreadData;

atomic_int active_threads_atomic = 0;

void *gauss_elimination_atomic(void *arg) {
    ThreadData *data = (ThreadData *)arg;
    int n = data->n;
    double **A = data->A;
    double *b = data->b;

    for (int k = 0; k < n; k++) {
        atomic_fetch_add(&active_threads_atomic, 1);

        for (int i = data->start; i < data->end; i++) {
            if (i > k) {
                double factor = A[i][k] / A[k][k];
                for (int j = k; j < n; j++) {
                    A[i][j] -= factor * A[k][j];
                }
                b[i] -= factor * b[k];
            }
        }

        atomic_fetch_sub(&active_threads_atomic, 1);
    }

    pthread_exit(NULL);
}
```

```c
void back_substitution(int n, double **A, double *b, double *x) {
    for (int i = n - 1; i >= 0; i--) {
        x[i] = b[i] / A[i][i];
        for (int j = i - 1; j >= 0; j--) {
            b[j] -= A[j][i] * x[i];
        }
    }
}

int main(int argc, char *argv[]) {
    if (argc < 3) {
        printf("Использование: %s <размер матрицы> <максимальное количество потоков>\n", argv[0]);
        exit(EXIT_FAILURE);
    }

    int n = atoi(argv[1]);
    int max_threads_atomic = atoi(argv[2]);

    double **A = malloc(n * sizeof(double *));
    double *b = malloc(n * sizeof(double));
    double *x = malloc(n * sizeof(double));
    for (int i = 0; i < n; i++) {
        A[i] = malloc(n * sizeof(double));
        for (int j = 0; j < n; j++) {
            A[i][j] = (double)(rand() % 100);
        }
        b[i] = (double)(rand() % 100);
    }

    pthread_t threads[MAX_THREADS];
    ThreadData thread_data[MAX_THREADS];
```

```c
    int chunk_size = n / max_threads_atomic;
    for (int i = 0; i < max_threads_atomic; i++) {
        thread_data[i].n = n;
        thread_data[i].A = A;
        thread_data[i].b = b;
        thread_data[i].start = i * chunk_size;
        thread_data[i].end = (i + 1) * chunk_size;
        pthread_create(&threads[i], NULL, gauss_elimination_atomic, &thread_data[i]);
    }

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

    back_substitution(n, A, b, x);

    printf("Решение системы:\n");
    for (int i = 0; i < n; i++) {
        printf("x[%d] = %f\n", i, x[i]);
    }

    for (int i = 0; i < n; i++) {
        free(A[i]);
    }
    free(A);
    free(b);
    free(x);

    return 0;
}
```

mutex.c:

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

#define MAX_THREADS 16

typedef struct {
    int n;
    double **A;
    double *b;
    int start, end;
} ThreadData;

pthread_mutex_t mutex;

void *gauss_elimination_mutex(void *arg) {
    ThreadData *data = (ThreadData *)arg;
    int n = data->n;
    double **A = data->A;
    double *b = data->b;

    for (int k = 0; k < n; k++) {
        for (int i = data->start; i < data->end; i++) {
            if (i > k) {
                double factor = A[i][k] / A[k][k];
                for (int j = k; j < n; j++) {
                    A[i][j] -= factor * A[k][j];
                }
                b[i] -= factor * b[k];
            }
        }
    }

    pthread_exit(NULL);
}
```

```c
void back_substitution(int n, double **A, double *b, double *x) {
    for (int i = n - 1; i >= 0; i--) {
        x[i] = b[i] / A[i][i];
        for (int j = i - 1; j >= 0; j--) {
            b[j] -= A[j][i] * x[i];
        }
    }
}

int main(int argc, char *argv[]) {
    if (argc < 3) {
        printf("Использование: %s <размер матрицы> <максимальное количество потоков>\n", argv[0]);
        exit(EXIT_FAILURE);
    }

    int n = atoi(argv[1]);
    int max_threads_mutex = atoi(argv[2]);

    if (pthread_mutex_init(&mutex, NULL) != 0) {
        perror("Ошибка инициализации мьютекса");
        exit(EXIT_FAILURE);
    }

    double **A = malloc(n * sizeof(double *));
    double *b = malloc(n * sizeof(double));
    double *x = malloc(n * sizeof(double));
    for (int i = 0; i < n; i++) {
        A[i] = malloc(n * sizeof(double));
        for (int j = 0; j < n; j++) {
            A[i][j] = (double)(rand() % 100);
        }
        b[i] = (double)(rand() % 100);
    }

    pthread_t threads[MAX_THREADS];
    ThreadData thread_data[MAX_THREADS];
```

```c
    int chunk_size = n / max_threads_mutex;
    for (int i = 0; i < max_threads_mutex; i++) {
        thread_data[i].n = n;
        thread_data[i].A = A;
        thread_data[i].b = b;
        thread_data[i].start = i * chunk_size;
        thread_data[i].end = (i + 1) * chunk_size;
        pthread_create(&threads[i], NULL, gauss_elimination_mutex, &thread_data[i]);
    }

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

    back_substitution(n, A, b, x);

    printf("Решение системы:\n");
    for (int i = 0; i < n; i++) {
        printf("x[%d] = %f\n", i, x[i]);
    }

    for (int i = 0; i < n; i++) {
        free(A[i]);
    }
    free(A);
    free(b);
    free(x);

    pthread_mutex_destroy(&mutex);

    return 0;
}
```

Протокол работы программы

Тестирование

```
victoria@victoria:~/laba/os/OSlabs/laba2$ gcc mutex.c
victoria@victoria:~/laba/os/OSlabs/laba2$ ./a.out 10 6
Решение системы:
x[0] = 62.003041
x[1] = 89.489587
x[2] = 116.858309
x[3] = -69.660630
x[4] = -138.368672
x[5] = -222.077356
x[6] = -9.268089
x[7] = 14.030800
x[8] = -14.285068
x[9] = 5.411765
```

Strace

victoria@victoria:~/laba/os/OSlabs/laba2$ strace ./a.out 10 6
execve("./a.out", ["./a.out", "10", "6"], 0x7fffe715fad0 /* 36 vars */) = 0

```
        brk(NULL)                               = 0x556de6fc4000
        arch_prctl(0x3001 /* ARCH_??? */, 0x7fff39966860) = -1 EINVAL (Invalid argument)
        mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1,
0) = 0x7f6fb888e000
        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=17839, ...}, AT_EMPTY_PATH) = 0
        mmap(NULL, 17839, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f6fb8889000
        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\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) =
0x7f6fb8660000
        mprotect(0x7f6fb8688000, 2023424, PROT_NONE) = 0
        mmap(0x7f6fb8688000, 1658880, PROT_READ|PROT_EXEC,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) = 0x7f6fb8688000
        mmap(0x7f6fb881d000, 360448, PROT_READ,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1bd000) = 0x7f6fb881d000
        mmap(0x7f6fb8876000, 24576, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x215000) = 0x7f6fb8876000
        mmap(0x7f6fb887c000, 52816, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7f6fb887c000
        close(3)                                = 0
        mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1,
0) = 0x7f6fb865d000
        arch_prctl(ARCH_SET_FS, 0x7f6fb865d740) = 0
        set_tid_address(0x7f6fb865da10)         = 208813
        set_robust_list(0x7f6fb865da20, 24)     = 0
        rseq(0x7f6fb865e0e0, 0x20, 0, 0x53053053) = 0
        mprotect(0x7f6fb8876000, 16384, PROT_READ) = 0
        mprotect(0x556de6408000, 4096, PROT_READ) = 0
        mprotect(0x7f6fb88c8000, 8192, PROT_READ) = 0
        prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY})
= 0
        munmap(0x7f6fb8889000, 17839)           = 0
        getrandom("\x2c\x33\x72\x06\x7c\x70\xa9\xa9", 8, GRND_NONBLOCK) = 8
        brk(NULL)                               = 0x556de6fc4000
        brk(0x556de6fe5000)                     = 0x556de6fe5000
        rt_sigaction(SIGRT_1, {sa_handler=0x7f6fb86f1870, sa_mask=[],
sa_flags=SA_RESTORER|SA_ONSTACK|SA_RESTART|SA_SIGINFO,
sa_restorer=0x7f6fb86a2520}, 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) = 0x7f6fb7e5c000
        mprotect(0x7f6fb7e5d000, 8388608, PROT_READ|PROT_WRITE) = 0
        rt_sigprocmask(SIG_BLOCK, ~[], [], 8)   = 0
        clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THR
EAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLE
ARTID, child_tid=0x7f6fb865c910, parent_tid=0x7f6fb865c910, exit_signal=0,
```

stack=0x7f6fb7e5c000, stack_size=0x7fff00, tls=0x7f6fb865c640} => {parent_tid=[208814]}, 88) = 208814

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

mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) = 0x7f6fb763b000

mprotect(0x7f6fb763c000, 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=0x7f6fb7e3b910, parent_tid=0x7f6fb7e3b910, exit_signal=0, stack=0x7f6fb763b000, stack_size=0x7fff00, tls=0x7f6fb7e3b640} => {parent_tid=[0]}, 88) = 208815

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

mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) = 0x7f6fb6e3a000

mprotect(0x7f6fb6e3b000, 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=0x7f6fb763a910, parent_tid=0x7f6fb763a910, exit_signal=0, stack=0x7f6fb6e3a000, stack_size=0x7fff00, tls=0x7f6fb763a640} => {parent_tid=[0]}, 88) = 208816

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

mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) = 0x7f6fb6639000

mprotect(0x7f6fb663a000, 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=0x7f6fb6e39910, parent_tid=0x7f6fb6e39910, exit_signal=0, stack=0x7f6fb6639000, stack_size=0x7fff00, tls=0x7f6fb6e39640} => {parent_tid=[0]}, 88) = 208817

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

mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) = 0x7f6fb5e38000

mprotect(0x7f6fb5e39000, 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=0x7f6fb6638910, parent_tid=0x7f6fb6638910, exit_signal=0, stack=0x7f6fb5e38000, stack_size=0x7fff00, tls=0x7f6fb6638640} => {parent_tid=[0]}, 88) = 208818

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

mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) = 0x7f6fb5637000

mprotect(0x7f6fb5638000, 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=0x7f6fb5e37910, parent_tid=0x7f6fb5e37910, exit_signal=0, stack=0x7f6fb5637000, stack_size=0x7fff00, tls=0x7f6fb5e37640} => {parent_tid=[0]}, 88) = 208819

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

munmap(0x7f6fb7e5c000, 8392704) = 0

munmap(0x7f6fb763b000, 8392704) = 0

newfstatat(1, "", {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0x4), ...}, AT_EMPTY_PATH) = 0

write(1, "\320\240\320\265\321\210\320\265\320\275\320\270\320\265\321\201\320\270\321\201\321\202\320\265\320\274\321\213:\n", 31Решение системы:

) = 31

```
write(1, "x[0] = 62.003041\n", 17x[0] = 62.003041
)      = 17
write(1, "x[1] = 89.489587\n", 17x[1] = 89.489587
)      = 17
write(1, "x[2] = 116.858309\n", 18x[2] = 116.858309
)     = 18
write(1, "x[3] = -69.660630\n", 18x[3] = -69.660630
)     = 18
write(1, "x[4] = -138.368672\n", 19x[4] = -138.368672
)    = 19
write(1, "x[5] = -222.077356\n", 19x[5] = -222.077356
)    = 19
write(1, "x[6] = -9.268089\n", 17x[6] = -9.268089
)      = 17
write(1, "x[7] = 14.030800\n", 17x[7] = 14.030800
)      = 17
write(1, "x[8] = -14.285068\n", 18x[8] = -14.285068
)     = 18
write(1, "x[9] = 5.411765\n", 16x[9] = 5.411765
)       = 16
exit_group(0)                           = ?
+++ exited with 0 +++
```

Вывод

В ходе выполнения лабораторной работы я освоила процесс создания многопоточных программ на языке Си, а также научилась синхронизировать потоки с использованием мьютексов. В процессе тестирования я проанализировала влияние количества потоков на производительность и ускорение выполнения алгоритма.