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-214БВ-24

Студент: Татульян А.Г.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата:
25.10.2025

Москва, 2025

Постановка задачи

Вариант 20.

Дан массив координат (x, y, z). Необходимо найти три точки, которые образуют треугольник максимальной площади.

Общий метод и алгоритм решения

Использованные системные вызовы:

- *ssize_t write(int fd, const void *buf, size_t count);* - записывает данные из буфера в файловый дескриптор.
- *void exit(int status);* - завершает выполнение процесса с указанным статусом.
- *int pthread_create(pthread_t *thread, const pthread_attr_t *attr, void *(*start_routine)(void *), void *arg);* - создаёт новый поток выполнения.
- *int pthread_join(pthread_t thread, void **thread_return);* - ожидает завершения выполнения переданного потока и записывает его возвращаемое значение в указатель.
- *int clock_gettime(clockid_t clock_id, timespec *spec);* - извлекает время из переданных часов.

Программа формирует произвольный массив точек. В одном потоке считается эталонный ответ на задачу, в соответствии с условием.

Для каждого числа заранее заданного количества потоков запускается в многопоточном режиме подсчёт ответа на задачу, проводится замер времени выполнения функции подсчёта и результат сравнивается на правильность с эталонным вариантом. Результаты всех метрик (время выполнения функции, ускорение и эффективность) выводятся в консоль.

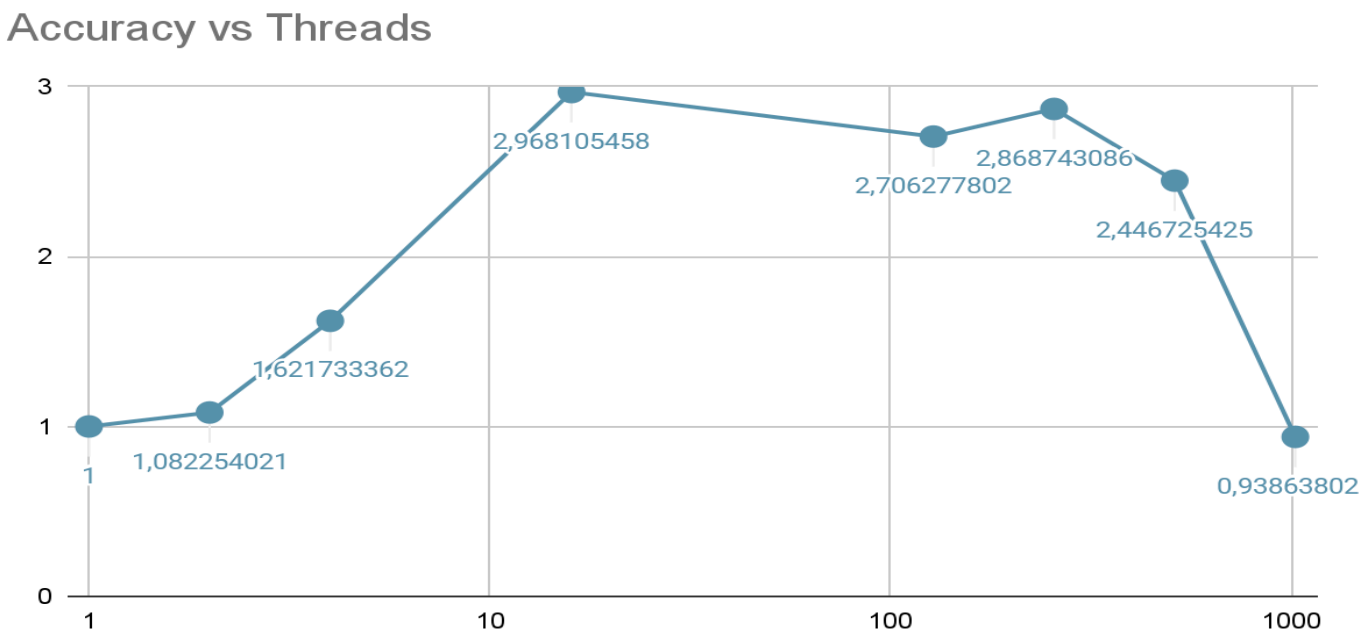
Поиск заданных точек производится перебором всех возможных треугольников из заданных точек. Равномерное разложение задачи по потокам осуществляется через разделение на равное количество обрабатываемых элементов по внешнему индексу обхода цикла.

Анализ метрик

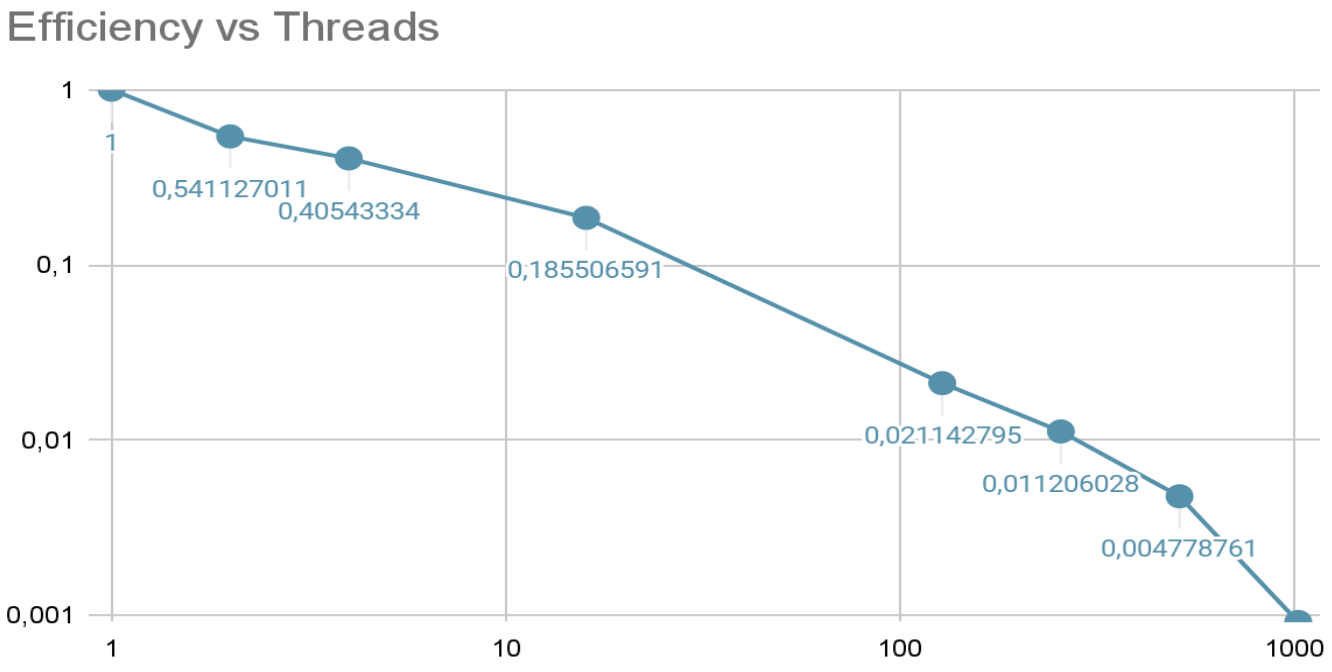
Число потоков	Время выполнения (мс)	Ускорение	Эффективность
1	5763.967091000	1.000000000	1.000000000
2	5325.891128000	1.082254021	0.541127011
4	3554.201466000	1.621733362	0.405433340
16	1941.968428000	2.968105458	0.185506591
128	2129.850486000	2.706277802	0.021142795
256	2009.230844000	2.868743086	0.011206028
512	2355.788284000	2.446725425	0.004778761
1024	6140.777343000	0.938638021	0.000916639

Ускорение показывает во сколько раз применение параллельного алгоритма уменьшает время решения задачи по сравнению с последовательным алгоритмом. Ускорение определяется величиной $S_N = T_1/T_N$, где T_1 - время выполнения на одном потоке, T_N - время выполнения на N потоках.

Эффективность - величина $E_N = S_N/N$, где S_N - ускорение, N - количество используемых потоков.



На графике зависимости ускорения от количества потоков видно, что максимальный эффект от параллелизма достигается при 16 потоках. Далее следует “плато”, а где-то после 256 потоков начинается падение, которое нивелирует весь положительный эффект от параллельного исполнения алгоритма.



По графику зависимости эффективности от количества потоков видно, что после 16 потоков начинаем стремительно терять эффективность, что соответствует графику зависимости ускорения от количества потоков, когда при 16 потоках достигается пик ускорения.

Оба графика демонстрируют, что существует ограничение роста производительности при увеличении числа вычислительных модулей. Поэтому хорошо видно, что постоянное линейное увеличение числа потоков может не только не помочь увеличить скорость, но и уменьшить её. Следует грамотно выбирать количество потоков, чтобы избежать этого эффекта.

Код программы

lab2.c

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>

#include <lab2/geometry.h>
#include <lab2/metrics.h>

#define POINTS_COUNT 1000
#define MAX_BUFFER_SIZE 1024

int main(void) {
    const uint64_t Threads[] = {
            1, 2, 4, 16, 128, 256, 512, 1024
    };

    const uint64_t ThreadsCount = sizeof(Threads) / sizeof(Threads[0]);

    Points points;
    points.data = (Point *) malloc(POINTS_COUNT * sizeof(Point));

    if (!points.data) {
        const char message[] = "Can`t allocate points!\n";

        write(STDOUT_FILENO, message, sizeof(message));

        return 1;
    }

    points.size = POINTS_COUNT;

    for (uint64_t i = 0; i < points.size; ++i) {
        points.data[i] = CreatePoint(rand() % 1000,
                                     rand() % 1000,
                                     rand() % 1000);
    }
```

```
    Triangle expected = MaxTriangle(points, 0);

    double linear = Measure(points,
Threads[0],                              expected);

    char buffer[MAX_BUFFER_SIZE + 1] = {0};
sprintf(buffer,
            "%.9f %.9f %.9f %lu\n",
linear,
            1.0,
            1.0,
            Threads[0]);     write(STDOUT_FILENO,
buffer, strlen(buffer));

    for (uint64_t i = 1; i < ThreadsCount; ++i) {
double total = Measure(points,
Threads[i],                                  expected);
double acceleration = Acceleration(linear,
total);         double efficiency =
Efficiency(acceleration,
Threads[i]);

        sprintf(buffer,
                "%.9f %.9f %.9f %lu\n",
total,                  acceleration,
efficiency,                  Threads[i]);
write(STDOUT_FILENO, buffer, strlen(buffer));     }

    free(points.data);

    return 0;
}
```

geometry.h

```
#ifndef GEOMETRY_H
#define GEOMETRY_H

#include <stdint.h>

typedef struct {
    double x, y, z;
} Point;
```

```
Point CreatePoint(double x,
                  double y,
                  double z);

typedef struct {
    Point *data;
    uint64_t size;
} Points;

typedef struct {
    Point a, b, c;

    double area;
} Triangle;

Triangle CreateTriangle(Point a,
                        Point b,
                        Point c,
                        double area);

double TriangleArea(Point a,
                    Point b,
                    Point c);

Triangle MaxTriangle(Points points,
                     uint64_t threads_count);

#endif
```

geometry.c

```
#include <math.h>
#include <pthread.h>
#include <stdbool.h>
#include <stdlib.h>
#include <unistd.h>

#include <lab2/geometry.h>

#define EPSILON 1e-9

Point CreatePoint(double x,
double y,                   double z) {
    Point point;
    point.x = x;
    point.y = y;
    point.z = z;

    return point;
```

```
}

Triangle CreateTriangle(Point a,
                        Point b,
                        Point c,
                        double area) {
    Triangle triangle;
    triangle.a = a;
    triangle.b = b;
    triangle.c = c;
    triangle.area = area;

    return triangle;
}

double TriangleArea(Point a,
                    Point b,
                    Point c) {
    double ab_x = b.x - a.x;
    double ab_y = b.y - a.y;
    double ab_z = b.z - a.z;

    double ac_x = c.x - a.x;
    double ac_y = c.y - a.y;
    double ac_z = c.z - a.z;

    double cross_x = ab_y * ac_z - ab_z * ac_y;
    double cross_y = ab_z * ac_x - ab_x * ac_z;
    double cross_z = ab_x * ac_y - ab_y * ac_x;

    double cross_length = sqrt(cross_x * cross_x + cross_y * cross_y + cross_z *
cross_z);

    return 0.5 * cross_length;
}

Triangle MaxTriangleLocal(Points points,
                          uint64_t start,
                          uint64_t end) {
    double local_max_area = 0.0;
    uint64_t max_a = 0, max_b = 0, max_c = 0;

    for (uint64_t i = start; i < end; ++i) {
        for (uint64_t j = i + 1; j < points.size; ++j) {
            for (uint64_t k = j + 1; k < points.size; ++k) {
                double triangle_area = TriangleArea(points.data[i],
                                                    points.data[j],
                                                    points.data[k]);
```

```
                if (triangle_area > local_max_area) {
                    local_max_area = triangle_area;
                    max_a = i;
                    max_b = j;
                    max_c = k;
                }
            }
        }
    }

    return CreateTriangle(points.data[max_a],
                          points.data[max_b],
                          points.data[max_c],
                          local_max_area);
}

typedef struct {
    Points points;

    uint64_t start, end;

    Triangle result;
} MaxTriangleLocalArgs;

void *MaxTriangleLocalThread(void *args) {
    MaxTriangleLocalArgs *arguments = (MaxTriangleLocalArgs *) args;

    arguments->result = MaxTriangleLocal(arguments->points,
                                         arguments->start,
                                         arguments->end);

    return NULL;
}

void JoinThreads(pthread_t *threads,
                 uint64_t threads_count) {
    for (uint64_t i = 0; i < threads_count; ++i) {
        if (pthread_join(threads[i], NULL) != 0) {
            const char message[] = "Can`t join thread!\n";

            write(STDOUT_FILENO, message, sizeof(message));
        }
    }
}

Triangle MaxTriangle(Points points,
                     uint64_t threads_count) {
    if (points.size < 3) {
        free(points.data);
```

```
        const char message[] = "Minimum count of points for calculate maximum triangle
area is 3!\n";

        write(STDOUT_FILENO, message, sizeof(message));

        exit(EXIT_FAILURE);
    }

    if (threads_count == 0) {
        return MaxTriangleLocal(points, 0, points.size);
    }

    uint64_t points_per_thread = points.size / threads_count;

    MaxTriangleLocalArgs *arguments = (MaxTriangleLocalArgs *) malloc(threads_count *
sizeof(MaxTriangleLocalArgs));

    if (!arguments) {
        free(points.data);

        const char message[] = "Can`t allocate memory for arguments!\n";

        write(STDOUT_FILENO, message, sizeof(message));

        exit(EXIT_FAILURE);
    }

    pthread_t *threads = malloc(threads_count * sizeof(pthread_t));

    if (!threads) {
        free(points.data);

        const char message[] = "Can`t allocate memory for threads!\n";

        write(STDOUT_FILENO, message, sizeof(message));

        exit(EXIT_FAILURE);
    }

    for (uint64_t i = 0, start = 0; i < threads_count; ++i, start += points_per_thread)
{
        arguments[i].points = points;
        arguments[i].start = start;
        arguments[i].end = start + points_per_thread;

        if (i + 1 == threads_count) {
            arguments[i].end = points.size;
        }
```

```
        if (pthread_create(&threads[i], NULL, MaxTriangleLocalThread, &arguments[i]) !=
0) {
            const char message_t[] = "Can`t create thread!\n";
            write(STDOUT_FILENO, message_t, sizeof(message_t));

            JoinThreads(threads, i);

            free(threads);
            free(arguments);

            free(points.data);

            exit(EXIT_FAILURE);
        }
    }

    bool joined = true;
    for (uint64_t i = 0; i < threads_count; ++i) {
        if (pthread_join(threads[i], NULL) != 0) {
            const char message[] = "Can`t join thread!\n";
            write(STDOUT_FILENO, message, sizeof(message));

            joined = false;
        }
    }

    free(threads);

    if (!joined) {
        free(arguments);
        free(points.data);

        exit(EXIT_FAILURE);
    }

    Triangle max_triangle = arguments[0].result;
    for (uint64_t i = 1; i < threads_count; ++i) {
        Triangle local_triangle = arguments[i].result;

        if (local_triangle.area - max_triangle.area > EPSILON) {
            max_triangle = local_triangle;
        }
    }

    free(arguments);

    return max_triangle;
}
```

metrics.h

```
#ifndef METRICS_H
#define METRICS_H

#include <lab2/geometry.h>


double Acceleration(double t_s,
double t_p);

double Efficiency(double s,                   uint64_t
p);

double Measure(Points points,                uint64_t
threads_count,                Triangle expected);

#endif //MAI_OS_2025_METRICS_H
```

metrics.c

```
#include <stdlib.h>
#include <time.h>
#include <unistd.h>

#include <lab2/metrics.h>


double Acceleration(double t_s,
double t_p) {
    return t_s / t_p;
}

double Efficiency(double s,                   uint64_t p)
{
    return s / (double) p;
}

double Measure(Points points,                uint64_t
threads_count,                Triangle expected) {
    struct timespec start, end;

    if (clock_gettime(CLOCK_MONOTONIC, &start) != 0) {
free(points.data);

        const char message[] = "Can`t get start time!\n";

        write(STDOUT_FILENO, message, sizeof(message));
```

```
            exit(EXIT_FAILURE);
        }

        Triangle calculated = MaxTriangle(points, threads_count);

        if (calculated.area != expected.area) {
            const char message[] = "[ERROR] Invalid value of calculation maximum area!\n";

            write(STDOUT_FILENO, message, sizeof(message));
        }

        if (clock_gettime(CLOCK_MONOTONIC, &end) != 0) {
            free(points.data);

            const char message[] = "Can`t get end time!\n";

            write(STDOUT_FILENO, message, sizeof(message));

            exit(EXIT_FAILURE);
        }

        return (double) (end.tv_sec - start.tv_sec) * 1000.0 + (double) (end.tv_nsec -
    start.tv_nsec) / 1000000.0;
    }
```

Протокол работы программы

Тестирование:

```
$ ./lab2
5763.967091000 1.000000000 1.000000000 1
5325.891128000 1.082254021 0.541127011 2
3554.201466000 1.621733362 0.405433340 4
1941.968428000 2.968105458 0.185506591 16
2129.850486000 2.706277802 0.021142795 128
2009.230844000 2.868743086 0.011206028 256
2355.788284000 2.446725425 0.004778761 512
6140.777343000 0.938638021 0.000916639 1024
```

Strace:

```
41620 execve("./lab2", ["./lab2"], 0x7ffc8e8b0188 /* 27 vars */) = 0
41620 brk(NULL)                           = 0x61bd954d6000
41620 mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS,
-1, 0) = 0x76d6fe907000
41620 access("/etc/ld.so.preload", R_OK) = -1 ENOENT (No such file or
directory)
41620 openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
41620 fstat(3, {st_mode=S_IFREG|0644, st_size=23159, ...}) = 0
```

```
     41620 mmap(NULL, 23159, PROT_READ, MAP_PRIVATE, 3, 0) = 0x76d6fe901000
     41620 close(3)                            = 0
     41620 openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libm.so.6",
O_RDONLY|O_CLOEXEC) = 3
     41620 read(3,
"\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\0\0\0\0\0\0\0\0"..., 832) =
832
     41620 fstat(3, {st_mode=S_IFREG|0644, st_size=952616, ...}) = 0
     41620 mmap(NULL, 950296, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) =
0x76d6fe818000
     41620 mmap(0x76d6fe828000, 520192, PROT_READ|PROT_EXEC,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x10000) = 0x76d6fe828000
     41620 mmap(0x76d6fe8a7000, 360448, PROT_READ,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x8f000) = 0x76d6fe8a7000
     41620 mmap(0x76d6fe8ff000, 8192, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0xe7000) = 0x76d6fe8ff000
     41620 close(3)                            = 0
     41620 openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6",
O_RDONLY|O_CLOEXEC) = 3
     41620 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\0"...,
832) = 832
     41620 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
     41620 fstat(3, {st_mode=S_IFREG|0755, st_size=2125328, ...}) = 0
     41620 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
     41620 mmap(NULL, 2170256, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) =
0x76d6fe600000
     41620 mmap(0x76d6fe628000, 1605632, PROT_READ|PROT_EXEC,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) = 0x76d6fe628000
     41620 mmap(0x76d6fe7b0000, 323584, PROT_READ,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1b0000) = 0x76d6fe7b0000
     41620 mmap(0x76d6fe7ff000, 24576, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1fe000) = 0x76d6fe7ff000
     41620 mmap(0x76d6fe805000, 52624, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x76d6fe805000
     41620 close(3)                            = 0
     41620 mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS,
-1, 0) = 0x76d6fe815000
     41620 arch_prctl(ARCH_SET_FS, 0x76d6fe815740) = 0
     41620 set_tid_address(0x76d6fe815a10)   = 41620
     41620 set_robust_list(0x76d6fe815a20, 24) = 0
     41620 rseq(0x76d6fe816060, 0x20, 0, 0x53053053) = 0
     41620 mprotect(0x76d6fe7ff000, 16384, PROT_READ) = 0
     41620 mprotect(0x76d6fe8ff000, 4096, PROT_READ) = 0
     41620 mprotect(0x61bd7b189000, 4096, PROT_READ) = 0
```

```
     41620 mprotect(0x76d6fe93f000, 8192, PROT_READ) = 0
     41620 prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024,
rlim_max=RLIM64_INFINITY}) = 0
     41620 munmap(0x76d6fe901000, 23159)     = 0
     41620 getrandom("\x4f\x0d\x2a\x94\xe5\xb7\xdf\xe2", 8, GRND_NONBLOCK) =
8
     41620 brk(NULL)                         = 0x61bd954d6000
     41620 brk(0x61bd954f7000)               = 0x61bd954f7000
     41620 rt_sigaction(SIGRT_1, {sa_handler=0x76d6fe699530, sa_mask=[],
sa_flags=SA_RESTORER|SA_ONSTACK|SA_RESTART|SA_SIGINFO,
sa_restorer=0x76d6fe645330}, NULL, 8) = 0
     41620 rt_sigprocmask(SIG_UNBLOCK, [RTMIN RT_1], NULL, 8) = 0
     41620 mmap(NULL, 8392704, PROT_NONE,
MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) = 0x76d6fddff000
     41620 mprotect(0x76d6fde00000, 8388608, PROT_READ|PROT_WRITE) = 0
     41620 rt_sigprocmask(SIG_BLOCK, ~[], [], 8) = 0
     41620
clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE
_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID,
child_tid=0x76d6fe5ff990, parent_tid=0x76d6fe5ff990, exit_signal=0,
stack=0x76d6fddff000, stack_size=0x7fff80, tls=0x76d6fe5ff6c0} =>
{parent_tid=[41621]}, 88) = 41621
     41621 rseq(0x76d6fe5fffe0, 0x20, 0, 0x53053053 <unfinished ...>
     41620 rt_sigprocmask(SIG_SETMASK, [],  <unfinished ...>
     41621 <... rseq resumed>)               = 0
     41620 <... rt_sigprocmask resumed>NULL, 8) = 0
     41621 set_robust_list(0x76d6fe5ff9a0, 24 <unfinished ...>
     41620 mmap(NULL, 8392704, PROT_NONE,
MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0 <unfinished ...>
     41621 <... set_robust_list resumed>)    = 0
     41620 <... mmap resumed>)               = 0x76d6fd5fe000
     41621 rt_sigprocmask(SIG_SETMASK, [],  <unfinished ...>
     41620 mprotect(0x76d6fd5ff000, 8388608, PROT_READ|PROT_WRITE <unfinished
...>
     41621 <... rt_sigprocmask resumed>NULL, 8) = 0
     41620 <... mprotect resumed>)           = 0
     41620 rt_sigprocmask(SIG_BLOCK, ~[], [], 8) = 0
     41620
clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE
_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID,
child_tid=0x76d6fddfe990, parent_tid=0x76d6fddfe990, exit_signal=0,
stack=0x76d6fd5fe000, stack_size=0x7fff80, tls=0x76d6fddfe6c0} =>
{parent_tid=[41622]}, 88) = 41622
     41622 rseq(0x76d6fddfefe0, 0x20, 0, 0x53053053 <unfinished ...>
     41620 rt_sigprocmask(SIG_SETMASK, [],  <unfinished ...>
     41622 <... rseq resumed>)               = 0
     41620 <... rt_sigprocmask resumed>NULL, 8) = 0
     41622 set_robust_list(0x76d6fddfe9a0, 24 <unfinished ...>
     41620 mmap(NULL, 8392704, PROT_NONE,
```

```
MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0 <unfinished ...>
     41622 <... set_robust_list resumed>)    = 0
     41620 <... mmap resumed>)               = 0x76d6fcdfd000
     41622 rt_sigprocmask(SIG_SETMASK, [],  <unfinished ...>
     41620 mprotect(0x76d6fcdfe000, 8388608, PROT_READ|PROT_WRITE <unfinished
...>
     41622 <... rt_sigprocmask resumed>NULL, 8) = 0
     41620 <... mprotect resumed>)           = 0
     41620 rt_sigprocmask(SIG_BLOCK, ~[], [], 8) = 0
     41620
clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE
_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID,
child_tid=0x76d6fd5fd990, parent_tid=0x76d6fd5fd990, exit_signal=0,
stack=0x76d6fcdfd000, stack_size=0x7fff80, tls=0x76d6fd5fd6c0} =>
{parent_tid=[41623]}, 88) = 41623
     41620 rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
     41620 mmap(NULL, 8392704, PROT_NONE,
MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) = 0x76d6fc5fc000
     41620 mprotect(0x76d6fc5fd000, 8388608, PROT_READ|PROT_WRITE) = 0
     41620 rt_sigprocmask(SIG_BLOCK, ~[], [], 8) = 0
     41620
clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|C
LONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID,
child_tid=0x76d6fcdfc990, parent_tid=0x76d6fcdfc990, exit_signal=0,
stack=0x76d6fc5fc000, stack_size=0x7fff80, tls=0x76d6fcdfc6c0} =>
{parent_tid=[41624]}, 88) = 41624
     41620 rt_sigprocmask(SIG_SETMASK, [],  <unfinished ...>
     41624 rseq(0x76d6fcdfcfe0, 0x20, 0, 0x53053053 <unfinished ...>
     41620 <... rt_sigprocmask resumed>NULL, 8) = 0
     41624 <... rseq resumed>)               = 0
     41620 futex(0x76d6fe5ff990, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME,
41621, NULL, FUTEX_BITSET_MATCH_ANY <unfinished ...>
     41624 set_robust_list(0x76d6fcdfc9a0, 24) = 0
     41624 rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
     41623 rseq(0x76d6fd5fdfe0, 0x20, 0, 0x53053053) = 0
     41623 set_robust_list(0x76d6fd5fd9a0, 24) = 0
     41623 rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
     41624 rt_sigprocmask(SIG_BLOCK, ~[RT_1], NULL, 8) = 0
     41624 madvise(0x76d6fc5fc000, 8368128, MADV_DONTNEED) = 0
     41624 exit(0)                           = ?
     41624 +++ exited with 0 +++
     41623 rt_sigprocmask(SIG_BLOCK, ~[RT_1], NULL, 8) = 0
     41623 madvise(0x76d6fcdfd000, 8368128, MADV_DONTNEED) = 0
     41623 exit(0)                           = ?
     41623 +++ exited with 0 +++
     41622 rt_sigprocmask(SIG_BLOCK, ~[RT_1], NULL, 8) = 0
     41622 madvise(0x76d6fd5fe000, 8368128, MADV_DONTNEED) = 0
     41622 exit(0)                           = ?
     41622 +++ exited with 0 +++
```

```
41621 rt_sigprocmask(SIG_BLOCK, ~[RT_1], NULL, 8) = 0
41621 madvise(0x76d6fddff000, 8368128, MADV_DONTNEED) = 0
41621 exit(0)                           = ?
41620 <... futex resumed>)              = 0
41621 +++ exited with 0 +++
41620 write(1, "1529.875462000 1.000000000 1.000"..., 41) = 41
41620 exit_group(0)                     = ?
41620 +++ exited with 0 +++
```

Вывод

В ходе лабораторной работы были успешно применены основные системные вызовы ОС Linux для работы с потоками для решения задачи многопоточного вычисления ответа по заданному условию. Получены метрики, наглядно показывающие границы увеличения количества одновременно выполняемых потоков для эффективного решения одной задачи.

Столкнулся с трудностями разделения поставленной задачи на множество мелких задач, пригодных для параллельного выполнения.