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: Зайцев Т.И.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 16.11.24

Москва, 2025

Постановка задачи

Вариант 7.

Два человека играют в кости. Правила игры следующие: каждый игрок делает бросок 2-ух костей K раз; побеждает тот, кто выбросил суммарно большее количество очков. Задача программы экспериментально определить шансы на победу каждого из игроков. На вход программе подается K, какой сейчас тур, сколько очков суммарно у каждого из игроков и количество экспериментов, которые должна произвести программа

Использованные системные вызовы:

1. sysconf(_SC_NPROCESSORS_ONLN) - Получение количества доступных логических процессоров.
2. syscall(SYS_getrandom, &seed, sizeof(seed), 0) - Прямой системный вызов для получения криптографически безопасных случайных чисел из ядра Linux.
3. gettimeofday(&start_time, NULL) - Получение текущего времени с микросекундной точностью.
4. write(STDERR_FILENO, msg, sizeof(msg) - 1) - Прямой системный вызов для записи в файловые дескрипторы.
5. getpid() - Получение идентификатора текущего процесса (PID).

Общий метод и алгоритм решения

Входные данные: Программа принимает аргументы командной строки, определяющие параметры выполнения.

Основные этапы работы:

1. Программа запускается с указанием: использовать 12 потоков, делать по 5 дополнительных бросков кубиков за ход, начинать с 0 очков у обоих игроков, и провести 1 миллион экспериментов.

2. Программа определяет, что нужно создать 12 потоков. Она проверяет доступное количество логических ядер процессора и создает общую структуру данных для хранения результатов, защищенную мьютексом для безопасного доступа из разных потоков.

3. 1 миллион экспериментов делится между 12 потоками - примерно по 83 тысячи экспериментов на каждый поток. Подготавливаются индивидуальные задания для каждого потока.

4. Операционная система создает 12 рабочих потоков. Каждый поток

получает:

- Свой собственный генератор случайных чисел с уникальным начальным значением
- Свою порцию работы (83 тысячи экспериментов)
- Доступ к общим результатам через мьютекс

5. Все 12 потоков начинают работать одновременно. Каждый эксперимент в потоке:
   - Начинается с 0 очков у обоих игроков
   - Выполняется 5 раундов бросков кубиков для обоих игроков
   - Сравниваются итоговые очки
   - Победителю записывается победа в локальный счетчик потока
   - Все вычисления происходят в локальных переменных потоков без необходимости синхронизации.

6. Когда поток завершает свои 83 тысячи экспериментов:

   - Он пытается захватить мьютекс
   - Если мьютекс занят другим потоком - переходит в состояние ожидания
   - Когда получает мьютекс - атомарно добавляет свои накопленные победы к общим результатам
   - Освобождает мьютекс для других потоков

7. Главная программа ожидает завершения всех 12 потоков через механизм join. После этого вычисляет и выводит:
   - Статистику побед каждого игрока в процентах
   - Общее время выполнения
   - Количество использованных потоков

8. Программа освобождает память и уничтожает мьютекс.

Анализ ускорения и эффективности

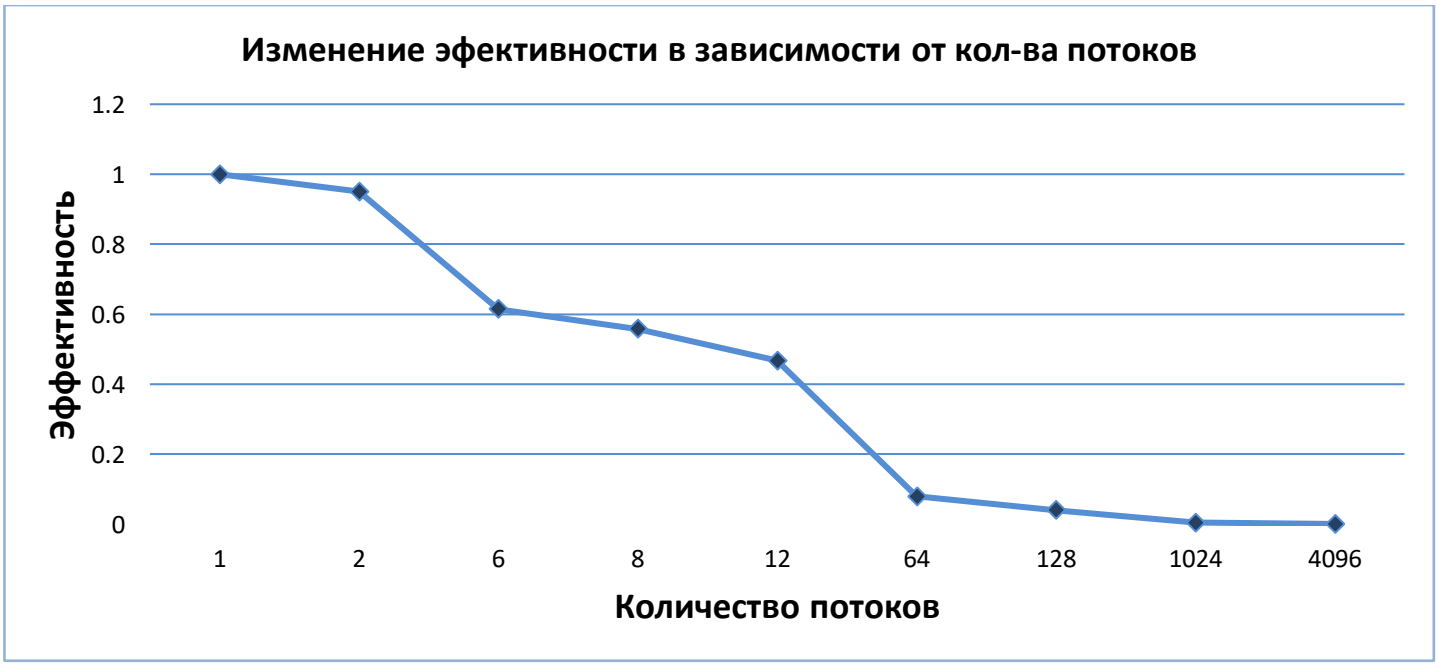
Ускорение - показывает, во сколько раз параллельная версия программы работает быстрее последовательной. Это основная метрика эффективности параллелизации.

$\mathbf{S_N = T_1/T_N}$, где T_1 – время выполнения на одном потоке, T_N – время выполнения на N потоках.

Эффективность - показывает, насколько хорошо используются вычислительные ресурсы. Определяет, какая доля времени потока тратится на полезную работу.

$\mathbf{E_N = S_N/N,}$ где S_N – ускорение, N – количество используемых потоков.

Число потоков	Время исполнения (мс)	Ускорение	Эффективность
1	1361	1	1
2	716	1,9	0,95
6	369	3,69	0,615
8	305	4,46	0,558
12	243	5,6	0,467
64	266	5,12	0,08
128	265	5,17	0,04
1024	276	4,93	0,005
4096	365	3,73	0,001



На графике зависимости ускорения от числа потоков четко видна оптимальная зона параллелизации при 12 потоках, где достигается максимальное ускорение 5.60. Это соответствует количеству логических процессоров в системе. При дальнейшем увеличении числа потоков наблюдается спад производительности - ускорение снижается до 3.73x при 4096 потоках.

График эффективности демонстрирует резкое падение после 12 потоков - с 47% до 0.1% при 4096 потоках. Наиболее эффективное использование ресурсов системы наблюдается при 2 потоках, где эффективность составляет 95%.

Полученные данные наглядно иллюстрируют закон Амдала, который ограничивает рост производительности при увеличении числа вычислителей. Для данного алгоритма оптимальным является использование 6-12 потоков, где достигается баланс между ускорением и эффективностью использования ресурсов.

Код программы

dice.h

```
#include <stdint.h>
#include <sys/syscall.h>
#include <unistd.h>
#include <pthread.h>
#include <sys/time.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

typedef struct {
    long wins1;
    long wins2;
    pthread_mutex_t mutex;
} shared_data_t;

typedef struct {
    int k;
    int player1_score;
    int player2_score;
    long experiments;
    int thread_id;
    shared_data_t* shared;
} thread_data_t;

int get_logical_cores();
static void init_prng();
static int rock_n_roll();
```

```
static unsigned int rand_xorshift();
void* thread_worker(void* arg);
void parallel_version(int k, int tour, int score1, int score2, long experiment, int
max_threads);
```

foo.c

```
#include "dice.h"

int get_logical_cores() {
    long cores = sysconf(_SC_NPROCESSORS_ONLN);
    if (cores <= 0) {
        return 4;
    }
    return (int)cores;
}

static __thread unsigned int seed = 0;

static void init_prng() {
    if (seed == 0) {
        // Simple and effective - try kernel RNG, fallback to time+PID
        if (syscall(SYS_getrandom, &seed, sizeof(seed), 0) != sizeof(seed)) {
            seed = (unsigned int)time(NULL) * (unsigned int)getpid();
        }
        if (seed == 0) seed = 1;
    }
}

static unsigned int rand_xorshift() {
    seed ^= seed << 13;
    seed ^= seed >> 17;
    seed ^= seed << 5;
    return seed;
}

static int rock_n_roll() {
    init_prng();
    return (rand_xorshift() % 6) + 1;
}


void* thread_worker(void* arg) {
    thread_data_t* data = (thread_data_t*)arg;
    long local_wins1 = 0, local_wins2 = 0;

    for (long exp = 0; exp < data->experiments; exp++) {
        int p1_total = data->player1_score;
        int p2_total = data->player2_score;

        for (int i = 0; i < data->k; i++) {
            p1_total += rock_n_roll() + rock_n_roll();
```

```
            p2_total += rock_n_roll() + rock_n_roll();
        }

        if (p1_total > p2_total) {
            local_wins1++;
        } else if (p2_total > p1_total) {
            local_wins2++;
        }
    }

    pthread_mutex_lock(&data->shared->mutex);
    data->shared->wins1 += local_wins1;
    data->shared->wins2 += local_wins2;
    pthread_mutex_unlock(&data->shared->mutex);

    return NULL;
}

void parallel_version(int k, int tour, int score1, int score2, long experiments, int
max_threads) {
    int actual_max_threads = max_threads > 0 ? max_threads : get_logical_cores();

    shared_data_t shared;
    shared.wins1 = 0;
    shared.wins2 = 0;

    if (pthread_mutex_init(&shared.mutex, NULL) != 0) {
        const char msg[] = "error: pthread_mutex_init failed\n";
        write(STDERR_FILENO, msg, sizeof(msg) - 1);
        return;
    }

    struct timeval start_time, end_time;
    gettimeofday(&start_time, NULL);

    int num_threads = (experiments < actual_max_threads) ? experiments : actual_max_threads;
    long experiments_per_thread = experiments / num_threads;
    long remaining_experiments = experiments % num_threads;

    pthread_t threads[num_threads];
    thread_data_t thread_data[num_threads];

    char info_msg[128];
    int info_len = snprintf(info_msg, sizeof(info_msg),
                            "Using %d threads (logical cores: %d)\n",
                            num_threads, get_logical_cores());
    write(STDERR_FILENO, info_msg, info_len);

    for (int i = 0; i < num_threads; i++) {
        thread_data[i].k = k;
        thread_data[i].player1_score = score1;
        thread_data[i].player2_score = score2;
        thread_data[i].experiments = experiments_per_thread;
        thread_data[i].thread_id = i;
        thread_data[i].shared = &shared;
```

```
        if (i < remaining_experiments) {
            thread_data[i].experiments++;
        }

        int result = pthread_create(&threads[i], NULL, thread_worker, &thread_data[i]);
        if (result != 0) {
            const char msg[] = "error: pthread_create failed\n";
            write(STDERR_FILENO, msg, sizeof(msg) - 1);

            for (int j = 0; j < i; j++) {
                pthread_cancel(threads[j]);
            }
            pthread_mutex_destroy(&shared.mutex);
            return;
        }
    }

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

    gettimeofday(&end_time, NULL);
    double execution_time = (end_time.tv_sec - start_time.tv_sec) +
                            (end_time.tv_usec - start_time.tv_usec) / 1000000.0;

    char msg[1024];
    int len;

    len = snprintf(msg, sizeof(msg), "Player 1 wins: %ld (%.2f%%)\n", shared.wins1, (dou-
ble)shared.wins1 / experiments * 100);
    write(STDOUT_FILENO, msg, len);

    len = snprintf(msg, sizeof(msg), "Player 2 wins: %ld (%.2f%%)\n", shared.wins2, (dou-
ble)shared.wins2 / experiments * 100);
    write(STDOUT_FILENO, msg, len);

    len = snprintf(msg, sizeof(msg), "Execution time: %.3f seconds\n", execution_time);
    write(STDOUT_FILENO, msg, len);

    len = snprintf(msg, sizeof(msg), "Threads used: %d\n", num_threads);
    write(STDOUT_FILENO, msg, len);

    pthread_mutex_destroy(&shared.mutex);
}
```

dice.c

```
#include "dice.h"

int main(int argc, char **argv) {
    int K, tour, pts1, pts2, max_threads;
```

```
    long exs;

    if (argc == 8 && strcmp(argv[1], "-m") == 0) {
        max_threads = atoi(argv[2]);
        K = atoi(argv[3]);
        tour = atoi(argv[4]);
        pts1 = atoi(argv[5]);
        pts2 = atoi(argv[6]);
        exs = atol(argv[7]);
    }
    else if (argc == 6) {
        K = atoi(argv[1]);
        tour = atoi(argv[2]);
        pts1 = atoi(argv[3]);
        pts2 = atoi(argv[4]);
        exs = atol(argv[5]);
        max_threads = get_logical_cores();
    }
    else {
        char msg[256];
        snprintf(msg, sizeof(msg), "Usage: %s [-m MAX_THREADS] K tour p1_pts p2_pts experi-
ments", argv[0]);
        write(STDERR_FILENO, msg, strlen(msg));
        return 1;
    }

    parallel_version(K, tour, pts1, pts2, exs, max_threads);
    return 0;
```

Протокол работы программы

```
tim@tim-potato-pc:~/OS_labs/lab_02/src$ ./game 5 1 0 0 10000000
Using 11 threads (logical cores: 11)
Player 1 wins: 4742200 (47.42%)
Player 2 wins: 4738530 (47.39%)
Execution time: 0.333 seconds
Threads used: 11
```

Вывод strace

```
tim@tim-potato-pc:~/OS_labs/lab_02/src$ strace -f ./game -m 2 5 1 0 0 1000000
execve("./game", ["./game", "-m", "2", "5", "1", "0", "0", "1000000"], 0x7ffd5a967680 /* 79
vars */) = 0
brk(NULL)                               = 0x63d64abef000
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x74fb266ce000
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (Нет такого файла или каталога)
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=69395, ...}) = 0
mmap(NULL, 69395, PROT_READ, MAP_PRIVATE, 3, 0) = 0x74fb266bd000
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) = 0x74fb26400000
mmap(0x74fb26428000, 1605632, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0x28000) = 0x74fb26428000
mmap(0x74fb265b0000, 323584, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1b0000) =
0x74fb265b0000
mmap(0x74fb265ff000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0x1fe000) = 0x74fb265ff000
mmap(0x74fb26605000, 52624, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0)
= 0x74fb26605000
close(3)                                = 0
mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x74fb266ba000
arch_prctl(ARCH_SET_FS, 0x74fb266ba740) = 0
set_tid_address(0x74fb266baa10)         = 95382
set_robust_list(0x74fb266baa20, 24)     = 0
rseq(0x74fb266bb060, 0x20, 0, 0x53053053) = 0
mprotect(0x74fb265ff000, 16384, PROT_READ) = 0
mprotect(0x63d62117f000, 4096, PROT_READ) = 0
mprotect(0x74fb2670c000, 8192, PROT_READ) = 0
prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0
munmap(0x74fb266bd000, 69395)           = 0
openat(AT_FDCWD, "/sys/devices/system/cpu/online", O_RDONLY|O_CLOEXEC) = 3
read(3, "0-11\n", 1024)                 = 5
close(3)                                = 0
write(2, "Using 2 threads (logical cores: "..., 36Using 2 threads (logical cores: 11)
) = 36
rt_sigaction(SIGRT_1, {sa_handler=0x74fb26499530, sa_mask=[],
sa_flags=SA_RESTORER|SA_ONSTACK|SA_RESTART|SA_SIGINFO, sa_restorer=0x74fb26445330}, 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) = 0x74fb25bff000
mprotect(0x74fb25c00000, 8388608, PROT_READ|PROT_WRITE) = 0
getrandom("\x5d\x1d\xb5\x22\xd3\x00\x91\x45", 8, GRND_NONBLOCK) = 8
brk(NULL)                               = 0x63d64abef000
brk(0x63d64ac10000)                     = 0x63d64ac10000
rt_sigprocmask(SIG_BLOCK, ~[], [], 8)   = 0
clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SE
TTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, child_tid=0x74fb263ff990, par-
ent_tid=0x74fb263ff990, exit_signal=0, stack=0x74fb25bff000, stack_size=0x7fff80,
tls=0x74fb263ff6c0}strace: Process 95383 attached
 => {parent_tid=[95383]}, 88) = 95383
[pid 95383] rseq(0x74fb263fffe0, 0x20, 0, 0x53053053 <unfinished ...>
[pid 95382] rt_sigprocmask(SIG_SETMASK, [],  <unfinished ...>
[pid 95383] <... rseq resumed>)         = 0
[pid 95382] <... rt_sigprocmask resumed>NULL, 8) = 0
[pid 95383] set_robust_list(0x74fb263ff9a0, 24 <unfinished ...>
[pid 95382] mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0 <unfin-
ished ...>
[pid 95383] <... set_robust_list resumed>) = 0
[pid 95382] <... mmap resumed>)         = 0x74fb253fe000
[pid 95383] rt_sigprocmask(SIG_SETMASK, [],  <unfinished ...>
[pid 95382] mprotect(0x74fb253ff000, 8388608, PROT_READ|PROT_WRITE <unfinished ...>
[pid 95383] <... rt_sigprocmask resumed>NULL, 8) = 0
```

```
[pid 95382] <... mprotect resumed>)     = 0
[pid 95383] getrandom( <unfinished ...>
[pid 95382] rt_sigprocmask(SIG_BLOCK, ~[],  <unfinished ...>
[pid 95383] <... getrandom resumed>"\x9b\x6c\xb1\x64", 4, 0) = 4
[pid 95382] <... rt_sigprocmask resumed>[], 8) = 0
[pid 95382]
clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SE
TTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, child_tid=0x74fb25bfe990, par-
ent_tid=0x74fb25bfe990, exit_signal=0, stack=0x74fb253fe000, stack_size=0x7fff80,
tls=0x74fb25bfe6c0}strace: Process 95384 attached
 => {parent_tid=[95384]}, 88) = 95384
[pid 95384] rseq(0x74fb25bfefe0, 0x20, 0, 0x53053053 <unfinished ...>
[pid 95382] rt_sigprocmask(SIG_SETMASK, [],  <unfinished ...>
[pid 95384] <... rseq resumed>)         = 0
[pid 95382] <... rt_sigprocmask resumed>NULL, 8) = 0
[pid 95384] set_robust_list(0x74fb25bfe9a0, 24 <unfinished ...>
[pid 95382] futex(0x74fb263ff990, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 95383, NULL,
FUTEX_BITSET_MATCH_ANY <unfinished ...>
[pid 95384] <... set_robust_list resumed>) = 0
[pid 95384] rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
[pid 95384] getrandom("\xc7\xc8\x22\xed", 4, 0) = 4
[pid 95383] rt_sigprocmask(SIG_BLOCK, ~[RT_1], NULL, 8) = 0
[pid 95383] madvise(0x74fb25bff000, 8368128, MADV_DONTNEED) = 0
[pid 95383] exit(0)                     = ?
[pid 95383] +++ exited with 0 +++
[pid 95382] <... futex resumed>)        = 0
[pid 95382] futex(0x74fb25bfe990, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 95384, NULL,
FUTEX_BITSET_MATCH_ANY <unfinished ...>
[pid 95384] rt_sigprocmask(SIG_BLOCK, ~[RT_1], NULL, 8) = 0
[pid 95384] madvise(0x74fb253fe000, 8368128, MADV_DONTNEED) = 0
[pid 95384] exit(0)                     = ?
[pid 95382] <... futex resumed>)        = 0
[pid 95384] +++ exited with 0 +++
write(1, "Player 1 wins: 473685 (47.37%)\n", 31Player 1 wins: 473685 (47.37%)
) = 31
write(1, "Player 2 wins: 474428 (47.44%)\n", 31Player 2 wins: 474428 (47.44%)
) = 31
write(1, "Execution time: 0.133 seconds\n", 30Execution time: 0.133 seconds
) = 30
write(1, "Threads used: 2\n", 16Threads used: 2
)       = 16
exit_group(0)                           = ?
+++ exited with 0 +++
```

Вывод

В ходе выполнения лабораторной работы были успешно изучены и применены основные системные вызовы POSIX Threads для создания многопоточной программы. Реализованная программа демонстрирует эффективное распараллеливание вычислительной задачи с достижением ускорения до 5.60x при использовании 12 потоков.

Экспериментальные результаты подтверждают теоретические положения о параллельных вычислениях: существует оптимальное количество потоков, при котором

достигается максимальное ускорение, а дальнейшее увеличение числа потоков приводит к росту накладных расходов и снижению эффективности. Для данной задачи оптимальным является использование 6-12 потоков, что соответствует количеству логических процессоров в тестовой системе.