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-216Б-23

Студент: Громова В.А.

Преподаватель:Бахарев В.Д.

Оценка: _____

Дата: 10.10.24

Москва, 2024

Постановка задачи

Вариант 7.

Родительский процесс создает дочерний процесс. Первой строчкой пользователь в консоль родительского процесса вводит имя файла, которое будет использовано для открытия файла с таким именем на чтение. Стандартный поток ввода дочернего процесса переопределяется открытым файлом. Дочерний процесс читает команды из стандартного потока ввода. Стандартный поток вывода дочернего процесса перенаправляется в pipe1. Родительский процесс читает из pipe1 и прочитанное выводит в свой стандартный поток вывода. Родительский и дочерний процесс должны быть представлены разными программами. В файле записаны команды вида: «число число число<endline>». Дочерний процесс считает их сумму и выводит результат в стандартный поток вывода. Числа имеют тип float. Количество чисел может быть произвольным.

Общий метод и алгоритм решения

В рамках лабораторной работы была разработана программа, способная обрабатывать данные в многопоточном режиме. Программа создает матрицу коэффициентов заданного размера, заполняет ее псевдослучайными числами с диагональным преобладанием и выполняет решение системы линейных уравнений методом Гаусса двумя способами, для сравнения последовательного и параллельного алгоритмов решения СЛАУ.
Сначала работает последовательная версия, которая в одном потоке выполняет полный алгоритм Гаусса - проходит по всей матрице, выполняя прямой и обратный ход для нахождения решений системы уравнений.
Затем запускается параллельная версия, где матрица разделяется между несколькими потоками - каждый поток обрабатывает свой участок строк матрицы и выполняет исключение переменных для своего блока строк, после чего главный поток собирает все результаты и определяет окончательные решения системы.
Программа выводит время работы каждого способа обработки, что демонстрирует ускорение работы при использовании многопоточности и позволяет исследовать зависимость производительности от количества потоков и размера матрицы.

Использованные системные вызовы и функции:

Функции управления временем:

gettimeofday(struct timeval *tv, struct timezone *tz) - получение текущего времени с микросекундной точностью для измерения производительности

Функции управления памятью:

void *malloc(size_t size) - динамическое выделение памяти для матрицы коэффициентов и вектора решений

void free(void *ptr) - освобождение выделенной памяти после использования

Функции ввода-вывода:

ssize_t write(int fd, const void *buf, size_t count) - вывод результатов и диагностической информации в стандартный поток вывода

int snprintf(char *str, size_t size, const char *format, ...) - безопасное форматирование строк для вывода

Функции многопоточности (POSIX Threads):

int pthread_create(pthread_t *thread, const pthread_attr_t *attr, void *(*start_routine) (void *), void *arg) - создание нового потока выполнения

int pthread_join(pthread_t thread, void **retval) - ожидание завершения работы потока и освобождение его ресурсов

int pthread_mutex_lock(pthread_mutex_t *mutex) - блокировка мьютекса для обеспечения эксклюзивного доступа к разделяемым данным

int pthread_mutex_unlock(pthread_mutex_t *mutex) - разблокировка мьютекса, позволяющая другим потокам получить доступ

Математические функции:

double fabs(double x) - вычисление абсолютного значения вещественного числа (используется при выборе ведущего элемента)

void srand(unsigned int seed) - инициализация генератора псевдослучайных чисел

int rand(void) - генерация последовательности псевдослучайных чисел для заполнения матрицы

Код программы

main.c

```c
#include <pthread.h>

#include <time.h>

#include <unistd.h>

#include <sys/time.h>

#include <stdlib.h>

#include <stdio.h>

#include <math.h>
```

```c
#include <string.h>


pthread_mutex_t mtx = PTHREAD_MUTEX_INITIALIZER;

int ready = 0, step = 0;

double **A = NULL, *X = NULL;

int N = 0, T = 0;


typedef struct {

   int id;

   int from;

   int to;

} thr_t;


long long now() {

   struct timeval tv;

   gettimeofday(&tv, NULL);

   return tv.tv_sec * 1000000LL + tv.tv_usec;

}


double** alloc(int n) {

   double **a = malloc(n * sizeof(double*));

   for (int i = 0; i < n; i++) a[i] = malloc((n+1) * sizeof(double));

   return a;

}


void free_mat(double **a, int n) {

   for (int i = 0; i < n; i++) free(a[i]);

   free(a);

}


void rand_mat(double **a, int n) {
```

```
    srand(time(NULL));
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            a[i][j] = (rand() % 1000) / 100.0;
            if (i == j) a[i][j] += 50.0;
        }
        a[i][n] = (rand() % 1000) / 100.0;
    }
}

void wait_all() {
    pthread_mutex_lock(&mtx);
    ready++;
    if (ready == T) {
        ready = 0;
        step++;
    } else {
        int cur = step;
        while (cur == step) pthread_mutex_unlock(&mtx), pthread_mutex_lock(&mtx);
    }
    pthread_mutex_unlock(&mtx);
}

void seq() {
    double **a = alloc(N);
    for (int i = 0; i < N; i++)
        for (int j = 0; j <= N; j++)
            a[i][j] = A[i][j];

    for (int k = 0; k < N; k++) {
        int mr = k;
```

```
        for (int i = k+1; i < N; i++)
            if (fabs(a[i][k]) > fabs(a[mr][k])) mr = i;

        if (mr != k)
            for (int j = k; j <= N; j++) {
                double t = a[k][j];
                a[k][j] = a[mr][j];
                a[mr][j] = t;
            }

        for (int j = k; j <= N; j++) a[k][j] /= a[k][k];

        for (int i = k+1; i < N; i++) {
            double f = a[i][k];
            for (int j = k; j <= N; j++) a[i][j] -= f * a[k][j];
        }
    }

    X = malloc(N * sizeof(double));
    for (int i = N-1; i >= 0; i--) {
        X[i] = a[i][N];
        for (int j = i+1; j < N; j++) X[i] -= a[i][j] * X[j];
        X[i] /= a[i][i];
    }

    free_mat(a, N);
}

void* par(void* arg) {
    thr_t *d = (thr_t*)arg;
```

```
for (int k = 0; k < N-1; k++) {

    if (d->id == 0) {

        pthread_mutex_lock(&mtx);

        int mr = k;

        for (int i = k+1; i < N; i++)

            if (fabs(A[i][k]) > fabs(A[mr][k])) mr = i;



        if (mr != k)

            for (int j = k; j <= N; j++) {

                double t = A[k][j];

                A[k][j] = A[mr][j];

                A[mr][j] = t;

            }

        pthread_mutex_unlock(&mtx);

    }



    wait_all();



    if (d->id == 0) {

        double div = A[k][k];

        for (int j = k; j <= N; j++) A[k][j] /= div;

    }



    wait_all();



    for (int i = d->from; i < d->to; i++) {

        if (i > k) {

            double f = A[i][k];

            for (int j = k; j <= N; j++) A[i][j] -= f * A[k][j];

        }

    }
```

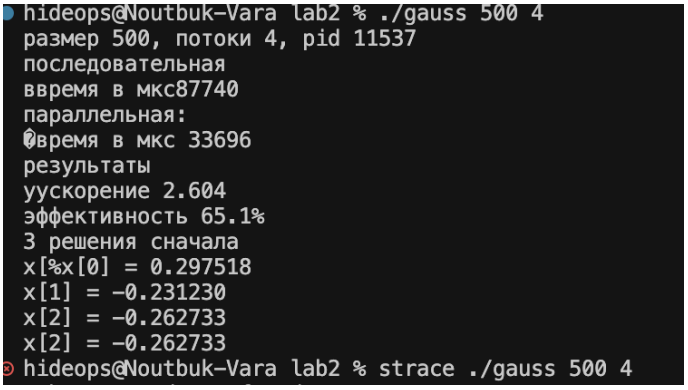
```
        wait_all();
    }

    wait_all();

    if (d->id == 0) {
        X = malloc(N * sizeof(double));
        for (int i = N-1; i >= 0; i--) {
            X[i] = A[i][N];
            for (int j = i+1; j < N; j++) X[i] -= A[i][j] * X[j];
            X[i] /= A[i][i];
        }
    }

    wait_all();

    return NULL;
}

int main(int argc, char* argv[]) {
    if (argc != 3) {
        write(STDOUT_FILENO, "usage ./program size threads\n",
              strlen("usage ./program size threads\n"));
        write(STDOUT_FILENO, "example ./program 500 4\n",
              strlen("example ./program 500 4\n"));
        return 1;
    }

    N = atoi(argv[1]);
    T = atoi(argv[2]);
```

```
    if (N <= 0 || T <= 0) return 1;
    if (T > N) T = N;

    char info[80];
    int len = snprintf(info, sizeof(info), "size %d, threads %d, pid %d\n", N, T,
getpid());
    write(STDOUT_FILENO, info, len);

    A = alloc(N);
    rand_mat(A, N);

    write(STDOUT_FILENO, "sequential \n", strlen("sequential \n"));

    long long t1 = now();
    seq();
    long long ts = now() - t1;
    snprintf(info, sizeof(info), "time  %lld microseconds\n", ts);
    write(STDOUT_FILENO, info, strlen(info));

    free(X);
    X = NULL;
    rand_mat(A, N);
    ready = step = 0;

    write(STDOUT_FILENO, "parallel \n", strlen("parallel \n"));

    pthread_t thr[T];
    thr_t data[T];
    int per = N / T, ext = N % T, cur = 0;
```

```
    for (int i = 0; i < T; i++) {
        data[i].id = i;
        data[i].from = cur;
        data[i].to = cur + per + (i < ext ? 1 : 0);
        cur = data[i].to;
    }

    t1 = now();
    for (int i = 0; i < T; i++) pthread_create(&thr[i], NULL, par, &data[i]);
    for (int i = 0; i < T; i++) pthread_join(thr[i], NULL);
    long long tp = now() - t1;

    snprintf(info, sizeof(info), "time %lld microseconds\n", tp);
    write(STDOUT_FILENO, info, strlen(info));

    double sp = (double)ts / tp;
    double ef = (sp / T) * 100.0;
    write(STDOUT_FILENO, "results \n", strlen("results \n"));
    snprintf(info, sizeof(info), "speedup  %.3f\nefficiency %.1f%%\n", sp, ef);
    write(STDOUT_FILENO, info, strlen(info));
    write(STDOUT_FILENO, "first 3 solutions \n", strlen("first 3 solutions \n"));
    for (int i = 0; i < 3 && i < N; i++) {
        snprintf(info, sizeof(info), "x[%d] = %.6f\n", i, X[i]);
        write(STDOUT_FILENO, info, strlen(info));
    }
    free_mat(A, N);
    free(X);
    write(STDOUT_FILENO, info, strlen(info));
    return 0;
}
```

Протокол работы программы

Вывод программы:



Тестирование:

./gauss 500 4

Strace:

```
execve("./gauss", ["./gauss", "500", "4"], 0x7ffe18158e70 /* 27 vars */) = 0
brk(NULL)                               = 0x5c2767343000
arch_prctl(0x3001 /* ARCH_??? */, 0x7ffe5259caf0) = -1 EINVAL (Invalid argument)
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7c1ee646e000
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=59280, ...}, AT_EMPTY_PATH) = 0
mmap(NULL, 59280, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7c1ee645f000
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\0O{\f\225\\=\201\327\312\301P\32$\230\266\235"..., 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) = 0x7c1ee6200000
mprotect(0x7c1ee6228000, 2023424, PROT_NONE) = 0
mmap(0x7c1ee6228000, 1658880, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0x28000) = 0x7c1ee6228000
mmap(0x7c1ee63bd000, 360448, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1bd000) =
0x7c1ee63bd000
mmap(0x7c1ee6416000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0x215000) = 0x7c1ee6416000
mmap(0x7c1ee641c000, 52816, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) =
0x7c1ee641c000
close(3)                                = 0
mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7c1ee645c000
arch_prctl(ARCH_SET_FS, 0x7c1ee645c740) = 0
set_tid_address(0x7c1ee645ca10)         = 9294
set_robust_list(0x7c1ee645ca20, 24)     = 0
rseq(0x7c1ee645d0e0, 0x20, 0, 0x53053053) = 0
mprotect(0x7c1ee6416000, 16384, PROT_READ) = 0
mprotect(0x5c274e319000, 4096, PROT_READ) = 0
mprotect(0x7c1ee64a8000, 8192, PROT_READ) = 0
prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0
```

```
munmap(0x7c1ee645f000, 59280)           = 0
getpid()                                = 9294
write(1, "size: 500, threads: 4, pid: 9294"..., 33) = 33
getrandom("\xb5\x49\x69\xe3\xfe\xc0\x11\x7f", 8, GRND_NONBLOCK) = 8
brk(NULL)                               = 0x5c2767343000
brk(0x5c2767364000)                     = 0x5c2767364000
brk(0x5c2767385000)                     = 0x5c2767385000
brk(0x5c27673a6000)                     = 0x5c27673a6000
brk(0x5c27673c7000)                     = 0x5c27673c7000
brk(0x5c27673e8000)                     = 0x5c27673e8000
brk(0x5c2767409000)                     = 0x5c2767409000
brk(0x5c276742a000)                     = 0x5c276742a000
brk(0x5c276744b000)                     = 0x5c276744b000
brk(0x5c276746c000)                     = 0x5c276746c000
brk(0x5c276748d000)                     = 0x5c276748d000
brk(0x5c27674ae000)                     = 0x5c27674ae000
brk(0x5c27674cf000)                     = 0x5c27674cf000
brk(0x5c27674f0000)                     = 0x5c27674f0000
brk(0x5c2767511000)                     = 0x5c2767511000
brk(0x5c2767532000)                     = 0x5c2767532000
write(1, "sequential:\n", 12)           = 12
brk(0x5c2767553000)                     = 0x5c2767553000
brk(0x5c2767574000)                     = 0x5c2767574000
brk(0x5c2767595000)                     = 0x5c2767595000
brk(0x5c27675b6000)                     = 0x5c27675b6000
brk(0x5c27675d7000)                     = 0x5c27675d7000
brk(0x5c27675f8000)                     = 0x5c27675f8000
brk(0x5c2767619000)                     = 0x5c2767619000
brk(0x5c276763a000)                     = 0x5c276763a000
brk(0x5c276765b000)                     = 0x5c276765b000
brk(0x5c276767c000)                     = 0x5c276767c000
brk(0x5c276769d000)                     = 0x5c276769d000
brk(0x5c27676be000)                     = 0x5c27676be000
brk(0x5c27676df000)                     = 0x5c27676df000
brk(0x5c2767700000)                     = 0x5c2767700000
brk(0x5c2767721000)                     = 0x5c2767721000
write(1, "time: 213666 microseconds\n", 26) = 26
brk(0x5c276754f000)                     = 0x5c276754f000
write(1, "parallel:\n", 10)             = 10
rt_sigaction(SIGRT_1, {sa_handler=0x7c1ee6291870, sa_mask=[],
sa_flags=SA_RESTORER|SA_ONSTACK|SA_RESTART|SA_SIGINFO, sa_restorer=0x7c1ee6242520}, 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) = 0x7c1ee59ff000
mprotect(0x7c1ee5a00000, 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_SETT
LS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, child_tid=0x7c1ee61ff910,
parent_tid=0x7c1ee61ff910, exit_signal=0, stack=0x7c1ee59ff000, stack_size=0x7fff00,
tls=0x7c1ee61ff640} => {parent_tid=[9295]}, 88) = 9295
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) = 0x7c1ee51fe000
mprotect(0x7c1ee51ff000, 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_SETT
LS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, child_tid=0x7c1ee59fe910,
```

```
parent_tid=0x7c1ee59fe910, exit_signal=0, stack=0x7c1ee51fe000, stack_size=0x7fff00,
tls=0x7c1ee59fe640} => {parent_tid=[9296]}, 88) = 9296
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) = 0x7c1ee49fd000
mprotect(0x7c1ee49fe000, 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_SETT
LS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, child_tid=0x7c1ee51fd910,
parent_tid=0x7c1ee51fd910, exit_signal=0, stack=0x7c1ee49fd000, stack_size=0x7fff00,
tls=0x7c1ee51fd640} => {parent_tid=[9297]}, 88) = 9297
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) = 0x7c1ee41fc000
mprotect(0x7c1ee41fd000, 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_SETT
LS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, child_tid=0x7c1ee49fc910,
parent_tid=0x7c1ee49fc910, exit_signal=0, stack=0x7c1ee41fc000, stack_size=0x7fff00,
tls=0x7c1ee49fc640} => {parent_tid=[9298]}, 88) = 9298
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
futex(0x7c1ee61ff910, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 9295, NULL,
FUTEX_BITSET_MATCH_ANY) = 0
write(1, "time: 330171 microseconds\n", 26) = 26
write(1, "results:\n", 9)               = 9
write(1, "speedup: 0.647\nefficiency: 16.2%"..., 33) = 33
write(1, "first 3 solutions:\n", 19)    = 19
write(1, "x[0] = 7.101914\n", 16)       = 16
write(1, "x[1] = 0.444875\n", 16)       = 16
write(1, "x[2] = 14.955520\n", 17)      = 17
getpid()                                = 9294
write(1, "\ncheck threads: ps -L -p 9294\n", 30) = 30
exit_group(0)                           = ?
+++ exited with 0 +++
```

число потоков	время мс	ускорение	эффективность
1	55.56	1.00	100.0
2	42.80	1.30	65.0
3	36.69	1.51	50.3
4	33.36	1.67	41.8
8	111.79	0.50	6.2
16	799.78	0.07	0.4

Ускорение показывает во сколько раз применение параллельного алгоритма уменьшает время решения задачи по сравнению с последовательным алгоритмом. Ускорение определяется величиной SN = T1/TN, где T1– время выполнения на одном потоке, TN – время выполнения на N потоках.

Эффективность – величина EN = SN/N, где SN – ускорение, N – количество используемых потоков.

График зависимости ускорения от количества потоков:

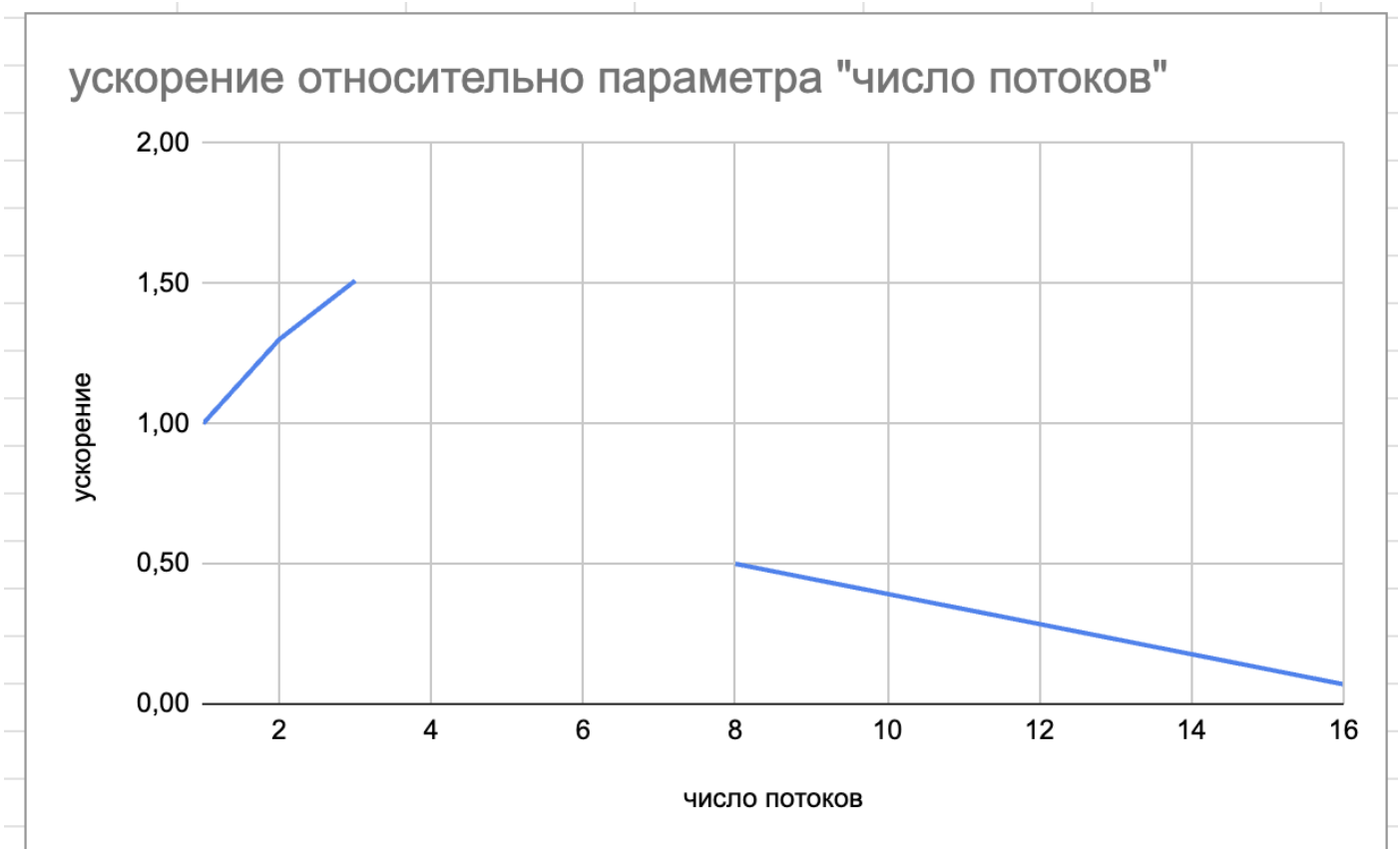
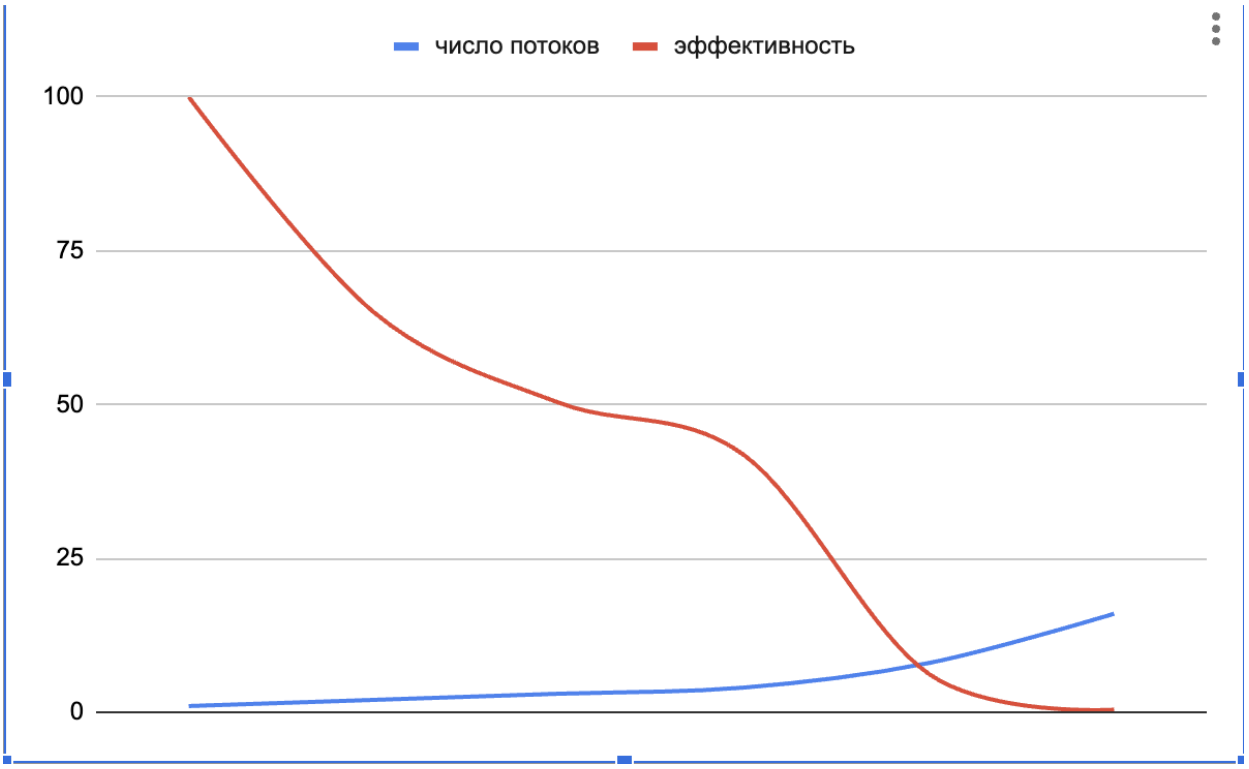


График зависимости эффективности от числа потоков:



Такие результаты характерны для алгоритмов с частой синхронизацией между потоками, каким является метод Гаусса. Каждый шаг прямого хода требует координации всех потоков через барьеры, что создает значительные накладные расходы:

На малых количествах потоков (2-4) выгода от параллелизма перевешивает затраты на синхронизацию, что дает положительное, хотя и далекое от идеального, ускорение.

При 8+ потоках затраты на создание потоков, синхронизацию и конкуренцию за ресурсы начинают превышать вычислительную выгоду. Особенно это заметно на относительно небольшой матрице 500×500, где объем работы на поток становится недостаточным для компенсации накладных расходов.

Низкая эффективность (падение с 65% до 0.4%) указывает на то, что большая часть процессорного времени тратится не на полезные вычисления, а на управление потоками, ожидание синхронизации и обновление общих переменных. Для алгоритма Гаусса оптимальным является использование количества потоков, не превышающего количество физических ядер процессора, поскольку дальнейшее увеличение приводит только к росту накладных расходов без реального выигрыша в производительности.

Вывод

В ходе лабораторной работы было установлено, что многопоточная реализация метода Гаусса обеспечивает умеренное ускорение вычислений при работе с матрицей размером 500×500. При этом оптимальная производительность достигается при 4 потоках с ускорением 1.67 раза.