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-24

Студент: Настина М.О.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 15.11.25

Москва, 2024

Постановка задачи

Вариант 1.

Отсортировать массив целых чисел при помощи битонической сортировки.

Общий метод и алгоритм решения

Использованные системные вызовы:

- *void malloc(size_t size); – выделяет динамическую память под массивы и временные структуры данных.
- *void free(void ptr); – освобождает ранее выделенную динамическую память.
- clock_t clock(void); – возвращает время процессора, использованное программой, для измерения производительности сортировки.
- **int pthread_create(pthread_t *thread, const pthread_attr_t *attr,
  void (start_routine)(void), void arg); – создает новые потоки выполнения для параллельной сортировки блоков массива.
- **int pthread_join(pthread_t thread, void retval); – ожидает завершения указанного потока и освобождает его ресурсы.
- **void *memcpy(void dest, const void src, size_t n); – копирует блоки памяти между исходным и временным массивами.
- pid_t getpid(void); – возвращает идентификатор текущего процесса (PID) для мониторинга.
- void srand(unsigned int seed); – инициализирует генератор псевдослучайных чисел для заполнения массива тестовыми данными.
- int rand(void); – генерирует псевдослучайные числа для заполнения массива.

Задачей было взять алгоритм битонической сортировки и запустить его в несколько потоков для ускорения работы кода. Алгоритм битонической сортировки состоит из части, которая сравнивает и меняет местами аргументы, находящиеся на определенном количестве «шагов» друг от друга, и части, которая разбивает массив на части и сортирует их в порядке возрастания или убывания. Программа работает следующим образом: создается массив случайных чисел, проходит проверка на степени двойки, массив распределяется на блоки по потокам, происходит сортировка и все блоки сливаются. Затем происходит проверка на корректность.

Код программы

main.c

```
#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#include <time.h>
```

```
#include <math.h>
#include <string.h>
#include <unistd.h>
#include <limits.h>

typedef struct {
    int *array;
    int low;
    int cnt;
    int dir;
} SortTask;

int MAX_THREADS;

void swap(int *a, int *b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}

void bitonicMerge(int arr[], int low, int cnt, int dir) {
    if (cnt > 1) {
        int k = cnt / 2;
        for (int i = low; i < low + k; i++) {
            if (dir == (arr[i] > arr[i + k])) {
                swap(&arr[i], &arr[i + k]);
            }
        }
        bitonicMerge(arr, low, k, dir);
        bitonicMerge(arr, low + k, k, dir);
```

```
    }
}

void bitonicSortSeq(int arr[], int low, int cnt, int dir) {
    if (cnt > 1) {
        int k = cnt / 2;
        bitonicSortSeq(arr, low, k, !dir);
        bitonicSortSeq(arr, low + k, k, dir);
        bitonicMerge(arr, low, cnt, dir);
    }
}

void* sortThread(void *arg) {
    SortTask *task = (SortTask *)arg;
    bitonicSortSeq(task->array, task->low, task->cnt, task->dir);
    free(task);
    return NULL;
}

void parallelBitonicSort(int *arr, int n, int dir) {
    int N = 1;
    while (N < n) N <<= 1;
    char buffer[100];
    snprintf(buffer, sizeof(buffer), "Исходный размер: %d, Приведен к: %d\n", n, N);
    write(STDOUT_FILENO, buffer, strlen(buffer));

    int *temp = malloc(N * sizeof(int));
    if (!temp) {
        write(STDOUT_FILENO, "err\n", 4);
        return;
```

```
}
memcpy(temp, arr, n * sizeof(int));
for (int i = n; i < N; i++) {
    temp[i] = INT_MAX;
}
if (N <= 1024 || MAX_THREADS == 1) {
    bitonicSortSeq(temp, 0, N, dir);
} else {
    int block_size = N / MAX_THREADS;
    while (block_size & (block_size - 1)) {
        block_size++;
    }
    if (block_size > N / 2) block_size = N / 2;
    pthread_t *threads = malloc(MAX_THREADS * sizeof(pthread_t));
    if (!threads) {
        write(STDOUT_FILENO, "err\n", 4);
        free(temp);
        return;
    }
    for (int i = 0; i < MAX_THREADS; i++) {
        SortTask *task = malloc(sizeof(SortTask));
        if (!task) {
            write(STDOUT_FILENO, "err\n", 4);
            continue;
        }
        task->array = temp;
        task->low = i * block_size;
        task->cnt = (i == MAX_THREADS - 1) ? (N - i * block_size) : block_size;
        task->dir = (i % 2 == 0) ? dir : !dir;
```

```
    if (pthread_create(&threads[i], NULL, sortThread, task) != 0) {
        char err_msg[20];
        snprintf(err_msg, sizeof(err_msg), "err %d\n", i);
        write(STDOUT_FILENO, err_msg, strlen(err_msg));
        free(task);
    }
}

for (int i = 0; i < MAX_THREADS; i++) {
    pthread_join(threads[i], NULL);
}
int merge_size = block_size;
while (merge_size < N) {
    for (int start = 0; start < N; start += 2 * merge_size) {
        int end = start + 2 * merge_size;
        if (end > N) end = N;
        int merge_dir = ((start / (2 * merge_size)) % 2 == 0) ? dir : !dir;
        for (int i = start; i < start + merge_size && i + merge_size < end; i++) {
            if (merge_dir == (temp[i] > temp[i + merge_size])) {
                swap(&temp[i], &temp[i + merge_size]);
            }
        }
        if (merge_size > 1) {
            for (int i = start; i < end; i += merge_size) {
                int block_end = i + merge_size;
                if (block_end > end) block_end = end;
                bitonicMerge(temp, i, block_end - i, merge_dir);
            }
        }
    }
```

```
            merge_size *= 2;
        }
        free(threads);
    }
    memcpy(arr, temp, n * sizeof(int));
    free(temp);
}

int isSorted(int *arr, int n) {
    for (int i = 1; i < n; i++) {
        if (arr[i] < arr[i - 1]) {
            char buffer[50];
            snprintf(buffer, sizeof(buffer), "err %d-%d: %d > %d\n",
                    i-1, i, arr[i-1], arr[i]);
            write(STDOUT_FILENO, buffer, strlen(buffer));
            return 0;
        }
    }
    return 1;
}

void testWithSmallArray() {
    write(STDOUT_FILENO, "\n=== ТЕСТ НА МАЛЕНЬКОМ МАССИВЕ ===\n", 37);
    int test_arr[] = {3, 7, 4, 8, 6, 2, 1, 5};
    int test_size = 8;
    write(STDOUT_FILENO, "Исходный массив: ", 18);
    for (int i = 0; i < test_size; i++) {
        char num[12];
        snprintf(num, sizeof(num), "%d ", test_arr[i]);
        write(STDOUT_FILENO, num, strlen(num));
```

```
    }
    write(STDOUT_FILENO, "\n", 1);
    parallelBitonicSort(test_arr, test_size, 1);
    write(STDOUT_FILENO, "Отсортированный массив: ", 24);
    for (int i = 0; i < test_size; i++) {
        char num[12];
        snprintf(num, sizeof(num), "%d ", test_arr[i]);
        write(STDOUT_FILENO, num, strlen(num));
    }
    write(STDOUT_FILENO, "\n", 1);
    if (isSorted(test_arr, test_size)) {
        write(STDOUT_FILENO, "Тест пройден\n", 13);
    } else {
        write(STDOUT_FILENO, "Тест не пройден\n", 16);
    }
}

int main(int argc, char *argv[]) {
    if (argc != 3) {
        char usage_msg[100];
        snprintf(usage_msg, sizeof(usage_msg),
                 "Использование: %s <количество_потоков> <размер_массива>\n",
                 argv[0]);
        write(STDOUT_FILENO, usage_msg, strlen(usage_msg));
        write(STDOUT_FILENO, "Пример: ", 8);
        write(STDOUT_FILENO, argv[0], strlen(argv[0]));
        write(STDOUT_FILENO, " 4 100000\n", 10);
        testWithSmallArray();
        return 1;
    }
```

```
MAX_THREADS = atoi(argv[1]);
int n = atoi(argv[2]);
if (MAX_THREADS < 1 || n < 1) {
    write(STDOUT_FILENO, "err\n", 4);
    return 1;
}
int *arr = malloc(n * sizeof(int));
if (!arr) {
    write(STDOUT_FILENO, "err\n", 4);
    return 1;
}
srand(time(NULL));
for (int i = 0; i < n; i++) {
    arr[i] = rand() % 1000000;
}
char info[100];
snprintf(info, sizeof(info), "Сортировка %d, потоков: %d\n", n, MAX_THREADS);
write(STDOUT_FILENO, info, strlen(info));
snprintf(info, sizeof(info), "PID: %d\n", getpid());
write(STDOUT_FILENO, info, strlen(info));
write(STDOUT_FILENO, "Первые 10: ", 11);
for (int i = 0; i < 10 && i < n; i++) {
    char num[12];
    snprintf(num, sizeof(num), "%d ", arr[i]);
    write(STDOUT_FILENO, num, strlen(num));
}
write(STDOUT_FILENO, "\n", 1);
clock_t start = clock();
parallelBitonicSort(arr, n, 1);
clock_t end = clock();
```

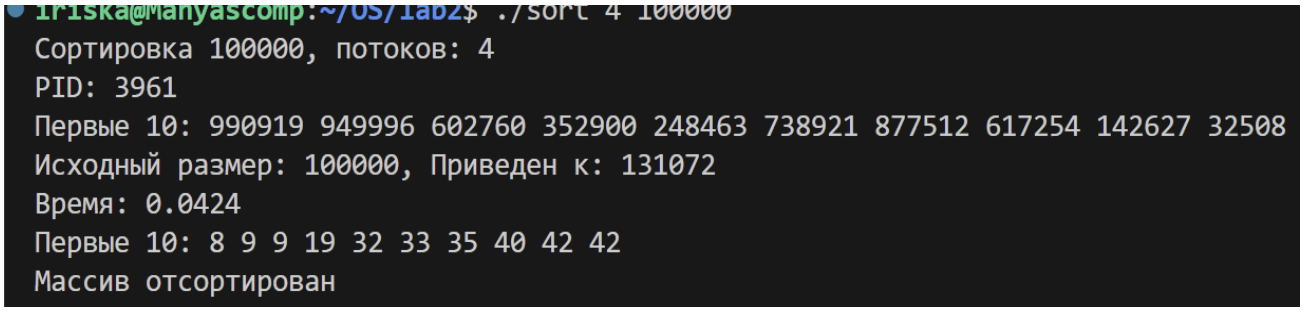
```
double time_taken = ((double)(end - start)) / CLOCKS_PER_SEC;
char time_msg[50];
snprintf(time_msg, sizeof(time_msg), "Время: %.4f \n", time_taken);
write(STDOUT_FILENO, time_msg, strlen(time_msg));
write(STDOUT_FILENO, "Первые 10: ", 11);
for (int i = 0; i < 10 && i < n; i++) {
    char num[12];
    snprintf(num, sizeof(num), "%d ", arr[i]);
    write(STDOUT_FILENO, num, strlen(num));
}
write(STDOUT_FILENO, "\n", 1);
if (isSorted(arr, n)) {
    write(STDOUT_FILENO, "Массив отсортирован\n", 20);
} else {
    write(STDOUT_FILENO, "Ошибка\n", 7);
}
write(STDOUT_FILENO, "\nИССЛЕДОВАНИЕ ПРОИЗВОДИТЕЛЬНОСТИ\n", 35);

int sizes[] = {1000, 10000, 100000, 1000000};
int num_sizes = 4;

for (int threads = 1; threads <= 8; threads *= 2) {
    char thread_msg[50];
    snprintf(thread_msg, sizeof(thread_msg), "\n--- %d поток(ов) ---\n", threads);
    write(STDOUT_FILENO, thread_msg, strlen(thread_msg));

    for (int s = 0; s < num_sizes; s++) {
        int n = sizes[s];
        int *arr = malloc(n * sizeof(int));
        srand(42);
```

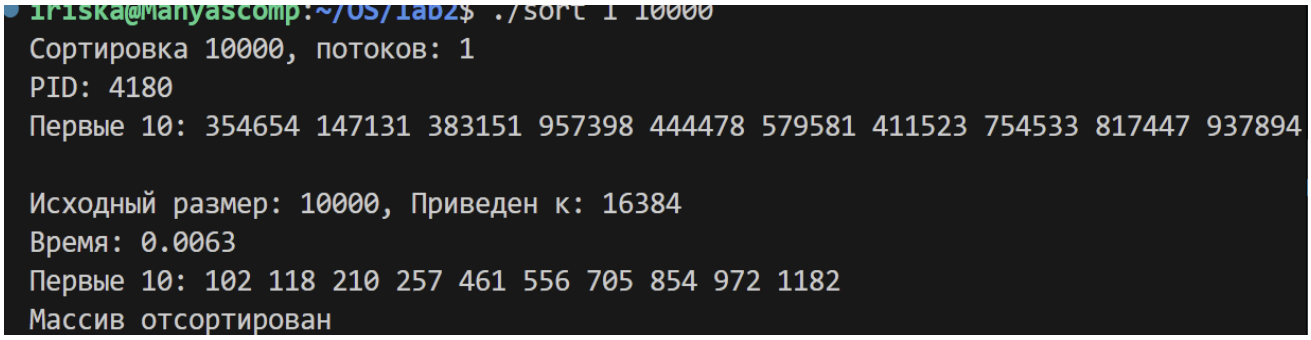
```
        for (int i = 0; i < n; i++) {
            arr[i] = rand() % 1000000;
        }

        MAX_THREADS = threads;
        clock_t start = clock();
        parallelBitonicSort(arr, n, 1);
        clock_t end = clock();

        double time_taken = ((double)(end - start)) / CLOCKS_PER_SEC;

        int sorted = 1;
        for (int i = 1; i < n; i++) {
            if (arr[i] < arr[i-1]) {
                sorted = 0;
                break;
            }
        }

        char result[100];
        snprintf(result, sizeof(result),
                 "Размер: %d, Время: %.4f сек, Корректность: %s\n",
                 n, time_taken, sorted ? "Да" : "Нет");
        write(STDOUT_FILENO, result, strlen(result));

        free(arr);
    }
}

free(arr);
```

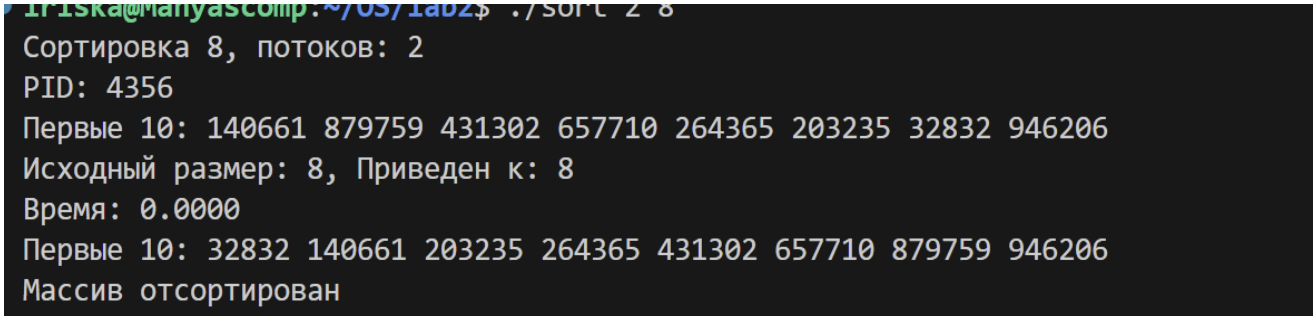
```
    return 0;
}
```

Протокол работы программы

Тестирование:







Strace:

clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, ...}, 88) = 10712

clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, ...}, 88) = 10713

clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, ...}, 88) = 10714

clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, ...}, 88) = 10715

clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, ...}, 88) = 10716

clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, ...}, 88) = 10717

clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, ...}, 88) = 10718

clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, ...}, 88) = 10719

clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, ...}, 88) = 10720

clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, ...}, 88) = 10721

clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, ...}, 88) = 10722

clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, ...}, 88) = 10723

clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, ...}, 88) = 10724

clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, ...}, 88) = 10725

clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, ...}, 88) = 10726

clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, ...}, 88) = 10727

clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, ...}, 88) = 10728

clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, ...}, 88) = 10729

clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, ...}, 88) = 10730

clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, ...}, 88) = 10731

clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, ...}, 88) = 10732

clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, ...}, 88) = 10733

clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, ...}, 88) = 10734

clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, ...}, 88) = 10735

clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, ...}, 88) = 10736

clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, ...}, 88) = 10737

clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, ...}, 88) = 10738

clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, ...}, 88) = 10739

clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, ...}, 88) = 10740

clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, ...}, 88) = 10741

clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, ...}, 88) = 10742

clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, ...}, 88) = 10743

clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, ...}, 88) = 10744

clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, ...}, 88) = 10745

clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, ...}, 88) = 10746

clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, ...}, 88) = 10747

clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, ...}, 88) = 10748

clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, ...}, 88) = 10749

clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, ...}, 88) = 10750

clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, ...}, 88) = 10751

clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, ...}, 88) = 10752

clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, ...}, 88) = 10753

clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, ...}, 88) = 10754

clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, ...}, 88) = 10755

clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, ...}, 88) = 10756

clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, ...}, 88) = 10757

Вывод

В рамках лабораторной работы была разработана и протестирована параллельная реализация алгоритма битонической сортировки. Программа демонстрирует различную эффективность в зависимости от размера обрабатываемых данных и количества используемых потоков. Наибольшее ускорение достигается при работе с крупными массивами, в то время как для небольших объемов данных многопоточность не дает преимуществ.

Количество потоков	Время, с	Ускорение	Эффективность
1	2.114	1.00	1
2	1.109	1.91	0.955
4	0.671	3.15	0.788
6	0.583	3.63	0.605
8	0.561	3.77	0.471
12	0.592	3.57	0.298
16	0.674	3.14	0.196
18	0.729	2.90	0.161

График зависимости ускорения от количества потоков:

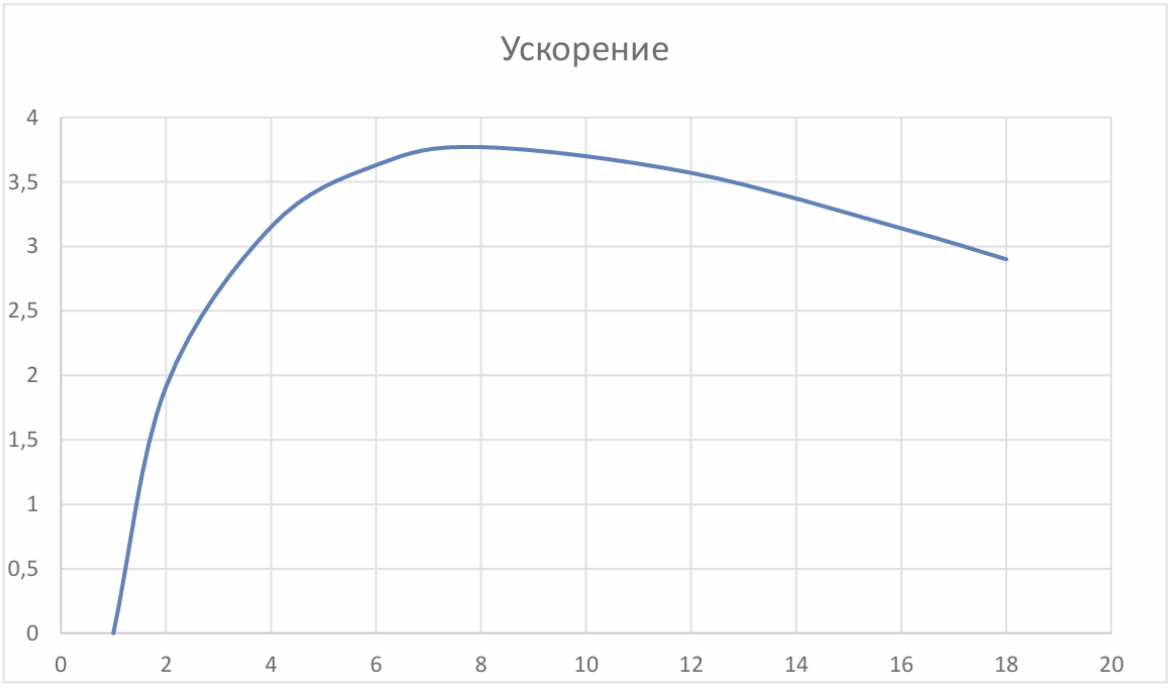
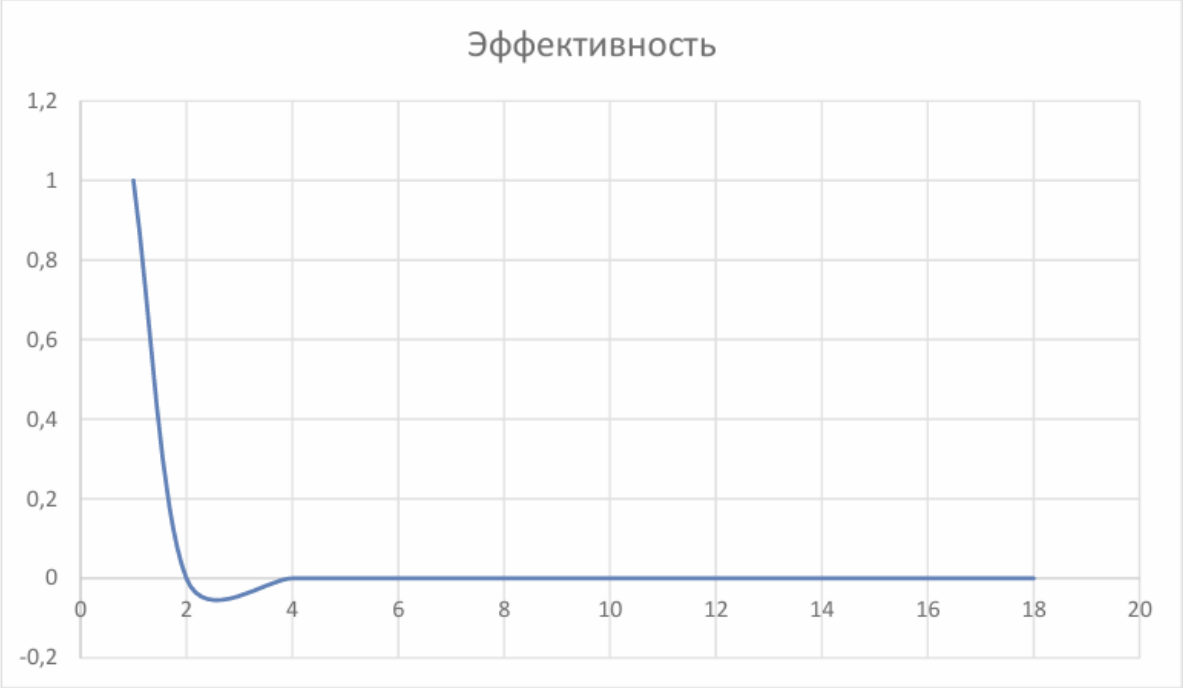


График зависимости эффективности от количества потоков:



Ускорение — это величина, которая показывает, во сколько раз параллельный алгоритм выполняется быстрее, чем последовательный.

Оно вычисляется по формуле:

$S_n = T_1 / T_n$,

где (T_1) — время работы на одном потоке, а (T_n) — время работы на (N) потоках.

Эффективность характеризует, насколько рационально используются дополнительные вычислительные ресурсы при распараллеливании.

Она определяется как:

$E_n = S_n / N$,

где (S_n) — ускорение, а (N) — количество потоков.