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-215Б-23

Студент: Агафонов А.С.

Преподаватель: Миронов Е.С.

Оценка: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата: 31.10.24

Москва, 2024

**Постановка задачи**

**Вариант 5.**

Отсортировать массив целых чисел при помощи четно-нечетной сортировки Бетчера.

**Общий метод и алгоритм решения**

Использованные системные вызовы:

* ifstream inputFile(const string &filename); - открытие файла для чтения.
* ofstream outputFile(const string &filename); - открытие файла для записи.
* std::thread - создание потока для выполнения функции сортировки.
* thread.join() - ожидание завершения выполнения потока.
* lock\_guard<mutex> lock(mtx); - синхронизация доступа к переменной swapped с помощью мьютекса.

В данной лабораторной работе мы реализуем алгоритм сортировки массива целых чисел с использованием параллельной четно-нечетной сортировки Бетчера. Для выполнения этой операции потребуется один массив чисел, который будет сортироваться.

Основная часть программы выполняется в вызывающем потоке и отвечает за ввод массива и инициализацию необходимых переменных.

**Четно-нечетная сортировка**

Для сортировки массива используется четно-нечетная сортировка (odd-even sort), которая выполняется в два этапа: четная и нечетная фазы. В четной фазе алгоритм сравнивает элементы массива с четными индексами с их последующими элементами, а в нечетной — элементы с нечетными индексами. Если пара соседних элементов не соответствует условию сортировки, они меняются местами. Алгоритм повторяется, пока массив не будет отсортирован.

**Многопоточность**

Для повышения производительности и эффективного использования ресурсов мы разделяем массив на части, которые обрабатываются параллельно несколькими потоками. Количество потоков задается параметром запуска программы <maxThreads>. Каждый поток отвечает за проверку и сортировку своей части массива.

Для синхронизации доступа к разделяемой переменной swapped, указывающей на необходимость дальнейшей сортировки, используется мьютекс mutex mtx. Завершение всех потоков на каждой фазе сортировки также контролируется с помощью метода join.

**Оптимизация использования памяти**

Для оптимизации использования памяти программа не создает дополнительные копии массива. Отсортированный массив сохраняется в тот же массив, который был прочитан из файла или введен через консоль.

Таким образом, эта программа предоставляет гибкий подход к сортировке с возможностью параллельного выполнения, что ускоряет процесс сортировки для больших массивов.

**Код программы**

#include <iostream>

#include <vector>

#include <thread>

#include <mutex>

#include <fstream>

#include <algorithm>

using namespace std;

mutex mtx; // Mutex для защиты переменной 'swapped'

void oddEvenSortParallel(vector<int>& arr, int maxThreads) {

    int n = arr.size();

    bool swapped = true;

    while (swapped) {

        swapped = false;

        // Две фазы: четная и нечетная

        for (int phase = 0; phase < 2; ++phase) {

            int threadsToUse = maxThreads;

            vector<thread> threads(threadsToUse);

            int chunkSize = (n + threadsToUse - 1) / threadsToUse;

            // Создаем потоки

            for (int t = 0; t < threadsToUse; ++t) {

                int startIndex = t \* chunkSize + phase;

                int endIndex = min((t + 1) \* chunkSize + phase, n - 1);

                if (startIndex >= n - 1) continue;

                threads[t] = thread([&, startIndex, endIndex]() {

                    bool localSwapped = false;

                    for (int i = startIndex; i < endIndex; i += 2) {

                        if (arr[i] > arr[i + 1]) {

                            swap(arr[i], arr[i + 1]);

                            localSwapped = true;

                        }

                    }

                    if (localSwapped) {

                        lock\_guard<mutex> lock(mtx);

                        swapped = true;

                    }

                });

            }

            // Ждем завершения потоков

            for (int t = 0; t < threadsToUse; ++t) {

                if (threads[t].joinable()) {

                    threads[t].join();

                }

            }

        }

    }

}

int main(int argc, char\* argv[]) {

    if (argc < 4) {

        cerr << "Usage: " << argv[0] << " <maxThreads> <inputFile> <outputFile>" << endl;

        return 1;

    }

    int maxThreads = stoi(argv[1]);

    string inputFileName = argv[2];

    string outputFileName = argv[3];

    vector<int> arr;

    // Чтение массива из файла

    ifstream inputFile(inputFileName);

    if (!inputFile) {

        cerr << "Error: Cannot open input file " << inputFileName << endl;

        return 1;

    }

    int num;

    while (inputFile >> num) {

        arr.push\_back(num);

    }

    inputFile.close();

    if (arr.empty()) {

        cerr << "Error: Input array is empty or invalid." << endl;

        return 1;

    }

    /\*

    cout << "Original array: ";

    for (int num : arr) {

        cout << num << " ";

    }

    cout << endl;

\*/

    oddEvenSortParallel(arr, maxThreads);

/\*

    cout << "Sorted array: ";

    for (int num : arr) {

        cout << num << " ";

    }

    cout << endl;

\*/

    // Запись отсортированного массива в файл

    ofstream outputFile(outputFileName);

    if (!outputFile) {

        cerr << "Error: Cannot open output file " << outputFileName << endl;

        return 1;

    }

    for (int num : arr) {

        outputFile << num << " ";

    }

    outputFile.close();

    return 0;

}

**Протокол работы программы**

**Тестирование:**

time ./sort 4 test.txt output.txt

real 0m23.202s

user 0m7.348s

sys 0m2.159s

**Демонстрация количества потоков:**

root@bdc60f1003f5:/workspaces/os\_base/lab2# ps -o nlwp 61127

NLWP

4

**Strace:**

execve("./sort", ["./sort"], 0x7fffa90ec4d8 /\* 30 vars \*/) = 0

brk(NULL) = 0x1b5b000

mmap(NULL, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7fa622042000

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=25258, ...}, AT\_EMPTY\_PATH) = 0

mmap(NULL, 25258, PROT\_READ, MAP\_PRIVATE, 3, 0) = 0x7fa62203b000

close(3) = 0

openat(AT\_FDCWD, "/usr/local/lib64/libstdc++.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\0\0\0\0\0\0\0\0"..., 832) = 832

newfstatat(3, "", {st\_mode=S\_IFREG|0755, st\_size=2530008, ...}, AT\_EMPTY\_PATH) = 0

mmap(NULL, 2543808, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0x7fa621dcd000

mmap(0x7fa621e72000, 1216512, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0xa5000) = 0x7fa621e72000

mmap(0x7fa621f9b000, 581632, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1ce000) = 0x7fa621f9b000

mmap(0x7fa622029000, 57344, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x25c000) = 0x7fa622029000

mmap(0x7fa622037000, 12480, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_ANONYMOUS, -1, 0) = 0x7fa622037000

close(3) = 0

openat(AT\_FDCWD, "/lib/x86\_64-linux-gnu/libm.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\0\0\0\0\0\0\0\0"..., 832) = 832

newfstatat(3, "", {st\_mode=S\_IFREG|0644, st\_size=907784, ...}, AT\_EMPTY\_PATH) = 0

mmap(NULL, 909560, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0x7fa621cee000

mmap(0x7fa621cfe000, 471040, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x10000) = 0x7fa621cfe000

mmap(0x7fa621d71000, 368640, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x83000) = 0x7fa621d71000

mmap(0x7fa621dcb000, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0xdc000) = 0x7fa621dcb000

close(3) = 0

openat(AT\_FDCWD, "/usr/local/lib64/libgcc\_s.so.1", O\_RDONLY|O\_CLOEXEC) = 3

read(3, "\177ELF\2\1\1\0\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

newfstatat(3, "", {st\_mode=S\_IFREG|0644, st\_size=906528, ...}, AT\_EMPTY\_PATH) = 0

mmap(NULL, 181160, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0x7fa621cc1000

mmap(0x7fa621cc5000, 143360, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x4000) = 0x7fa621cc5000

mmap(0x7fa621ce8000, 16384, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x27000) = 0x7fa621ce8000

mmap(0x7fa621cec000, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x2b000) = 0x7fa621cec000

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\20t\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

newfstatat(3, "", {st\_mode=S\_IFREG|0755, st\_size=1922136, ...}, 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, 1970000, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0x7fa621ae0000

mmap(0x7fa621b06000, 1396736, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x26000) = 0x7fa621b06000

mmap(0x7fa621c5b000, 339968, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x17b000) = 0x7fa621c5b000

mmap(0x7fa621cae000, 24576, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1ce000) = 0x7fa621cae000

mmap(0x7fa621cb4000, 53072, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_ANONYMOUS, -1, 0) = 0x7fa621cb4000

close(3) = 0

mmap(NULL, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7fa621ade000

arch\_prctl(ARCH\_SET\_FS, 0x7fa621adf480) = 0

set\_tid\_address(0x7fa621adf750) = 75309

set\_robust\_list(0x7fa621adf760, 24) = 0

rseq(0x7fa621adfda0, 0x20, 0, 0x53053053) = 0

mprotect(0x7fa621cae000, 16384, PROT\_READ) = 0

mprotect(0x7fa621cec000, 4096, PROT\_READ) = 0

mprotect(0x7fa621dcb000, 4096, PROT\_READ) = 0

mmap(NULL, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7fa621adc000

mprotect(0x7fa622029000, 45056, PROT\_READ) = 0

mprotect(0x409000, 4096, PROT\_READ) = 0

mprotect(0x7fa622074000, 8192, PROT\_READ) = 0

prlimit64(0, RLIMIT\_STACK, NULL, {rlim\_cur=8192\*1024, rlim\_max=RLIM64\_INFINITY}) = 0

munmap(0x7fa62203b000, 25258) = 0

futex(0x7fa62203773c, FUTEX\_WAKE\_PRIVATE, 2147483647) = 0

getrandom("\xb5\x0f\x29\xd3\x65\x36\xe2\xf5", 8, GRND\_NONBLOCK) = 8

brk(NULL) = 0x1b5b000

brk(0x1b7c000) = 0x1b7c000

write(2, "Usage: ", 7Usage: ) = 7

write(2, "./sort", 6./sort) = 6

write(2, " <maxThreads> <inputFile> <outpu"..., 38 <maxThreads> <inputFile> <outputFile>) = 38

write(2, "\n", 1

) = 1

exit\_group(1) = ?

+++ exited with 1 +++

Массив размерности 1000.

|  |  |  |  |
| --- | --- | --- | --- |
| Число потоков | Время исполнения (мс) | Ускорение | Эффективность |
| 1 | 80 | 1 | 1 |
| 2 | 120 | 0,6 | 0,3 |
| 3 | 136 | 0,588 | 0,196 |
| 4 | 164 | 0,488 | 0,121 |
| 5 | 235 | 0,34 | 0,068 |
| 6 | 241 | 0.332 | 0,006 |

Массив 60000.

|  |  |  |  |
| --- | --- | --- | --- |
| Число потоков | Время исполнения (мс) | Ускорение | Эффективность |
| 1 | 25144 | 1 | 1 |
| 2 | 15034 | 1.672 | 0.836 |
| 3 | 13264 | 1.896 | 0.632 |
| 4 | 14553 | 1.728 | 0.432 |
| 5 | 15481 | 1.624 | 0.325 |
| 6 | 15879 | 1.584 | 0.264 |

**Ускорение** показывает во сколько раз применение параллельного алгоритма уменьшает время решения задачи по сравнению с последовательным алгоритмом. Ускорение определяется величиной SN=T1/TN, где Т1 - время выполнения на одном потоке, TN - время выполнения на N потоках.

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

**Вывод**

На основе тестирования программы с разным количеством потоков и объемом данных можно сделать следующие выводы:

1. Многопоточность значительно ускоряет выполнение программы при грамотном распределении нагрузки между потоками.
2. Для максимального ускорения следует выбирать оптимальное количество потоков, которое соответствует вычислительным возможностям компьютера (например, числу ядер процессора) и объему задачи.
3. Избыточное количество потоков может снижать эффективность работы из-за накладных расходов на управление потоками и синхронизацию.

Таким образом, многопоточность является эффективным инструментом для повышения производительности, если её правильно применять в зависимости от аппаратных характеристик и сложности задачи.