Московский государственный университет имени М.В.Ломоносова

Факультет вычислительной математики и кибернетики

Кафедра системного программирования

Задание по курсу:

«Параллельные высокопроизводительные вычисления»

Отчёт по практическому заданию № 1:

«Расписание сети сортировки»

Арефьев Вениамин Андреевич

528 группа

Москва

11.11.2023

Содержание

Описание условий

Разработать последовательную программу вычисления расписания сети сортировки, числа использованных компараторов и числа тактов, необходимых для её срабатывания при выполнении на n процессорах. Число тактов сортировки при параллельной обработке не должно превышать числа тактов, затрачиваемых четно-нечетной сортировкой Бетчера.

Параметр командной строки запуска:

- n, где n>=1 – количество элементов в упорядочиваемом массиве, элементы которого расположены на строках с номерами [0…n-1]

Формат команды запуска:

bsort n

Требуется:

1. вывести в файл стандартного вывода расписание и его характеристики в представленном далее формате;
2. обеспечить возможность вычисления сети сортировки для числа элементов 1<=n<=10000;
3. предусмотреть полную проверку правильности сети сортировки для значений числа сортируемых элементов 1<=n<=24;
4. представить краткий отчет удовлетворяющий указанным далее требованиям.

Формат файла результата:

Начало файла результата

n 0 0

cu0 cd0

cu1 cd1

…

cun_comp-1 cdn_comp-1

n_comp

n_tact

Конец файла результата

Здесь:

- n 0 0 – число сортируемых элементов, ноль, ноль. Да, вывести число элементов и два нуля.
- cui cdi – номера строк, соединяемых i-м компаратором сравнения перестановки.
- n_comp – число компараторов
- n_tact – число тактов сети сортировкифывфыв

Описание метода решения

В ходе решения был выбрал и реализован рекурсивный алгоритм сортировки на основе четно-нечетного слияния Бэтчера. Общая схема работы проиллюстрирована на рисунках 1 и 2.

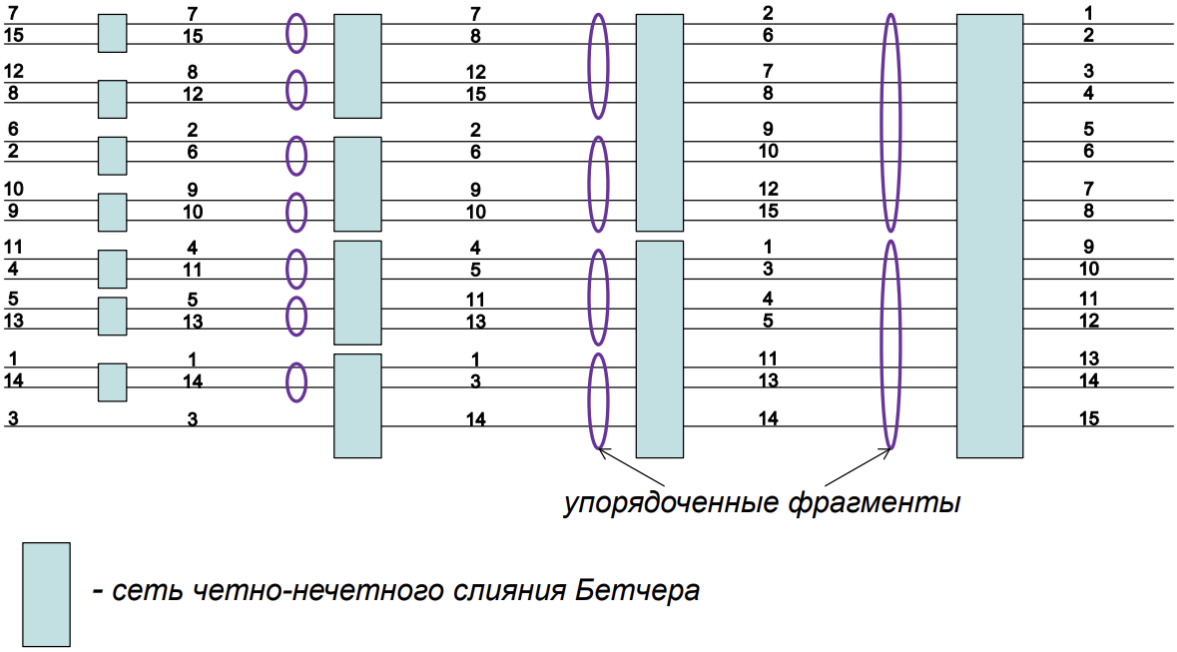

Рисунок 1. Сортировка массива из 15 элементов.

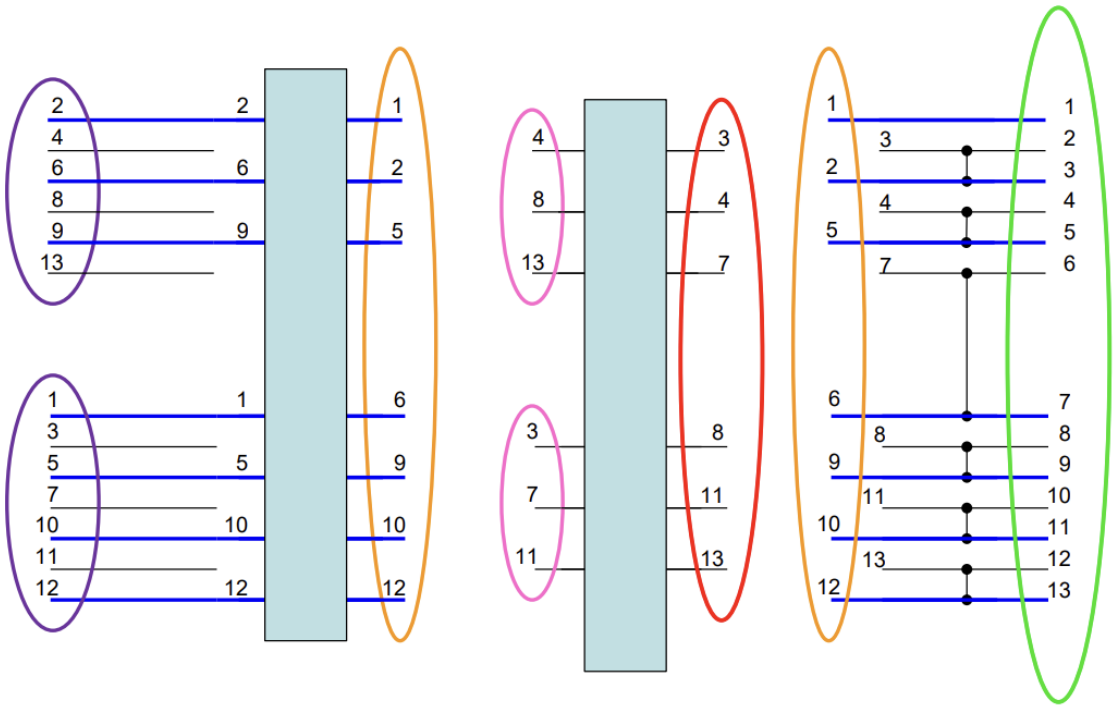

Рисунок 2. Сортировка массива из 13 элементов.

Работа алгоритма описывается следующими шагами:

1. Разделить исходный массив длины P на две части. Первая часть длины n =p/2, округленная вниз. А вторая часть длины m=p-n
2. Отсортировать получившиеся подмассивы, причём в каждой части происходит выделение чётных и нечётных элементов и отдельных сортировка этих пар
3. Произвести слияние отсортированных подмассивов с помощью сети чётно-нечётного слияния Бэтчера

В реализации первому пункту соответствует функция batcherSort, а второму и третьему – sortAndMergeTwoArrays.

Разработка велась на языке C++ с использованием стандарта C++20. Компиляция осуществлялась с помощью CMake, либо с помощью Makefile.

Описание метода проверки

Для запуска проверки с помощью 0–1 принципа необходимо вместо числа, как первого аргумента командной строки, подать строку «test». Тогда будут отсортированы всевозможные массивы длины n, где n принимает значения в диапазоне [1,24]. При правильной работе сортиров будет выведена 1, что она и вывела при запуске.

Для проверки правильности количества тактов и числа компараторов был написать скрипт, который запускает программу с аргументом n, который принимает значения в диапазоне [1,24]. Таким образом получается две последовательности – количество компараторов и тактов для каждой итерации. В таблицах 1, 2 и 3 можно увидеть количество тактов (cycles) и число использованных компараторов (comp) в зависимости от n, выведенные программой, а также аналитическую оценку число тактов (cycles_max).

Аналитическая оценка получена с помощью формулы: $\frac{\log_2 n(\log_2 n+1)}{2}$

n	1	2	3	4	5	6	7	8
comp	0	1	3	5	9	12	16	19
cycles	0	1	3	3	5	6	6	6
cycles_max	0	1	3	3	6	6	6	6

Таблица 1.

n	9	10	11	12	13	14	15	16
comp	26	31	37	41	48	53	59	63
cycles	8	9	10	10	10	10	10	10
cycles_max	10	10	10	10	10	10	10	10

Таблица 2.

N	17	18	19	20	21	22	23	24
Comp	74	82	91	97	107	114	122	127
Cycles	12	13	14	14	15	15	15	15
cycles_max	15	15	15	15	15	15	15	15

Таблица 3.

Как видно из таблицы, фактическое количество задействованных тактов не превосходит аналитической оценки, что подтверждает правильность алгоритма.

Приложение № 1: исходный текст программы

```
#include <iostream>
#include <cmath>
#include <vector>
#include <set>
#include <memory>

int comparator_count = 0;
bool is_printing = true;
std::vector<std::set<size_t> > compare_cycles;

enum {
    MAX_TEST_ARRAY_SIZE = 24
};

void addPairToCycles(size_t first, size_t second) {
    if (compare_cycles.empty()) {
        compare_cycles.push_back(std::set<size_t>{first, second});
        return;
    }
    int cycles_count = (int) compare_cycles.size();
    for (int i = cycles_count - 1; i >= 0; i--) {
        // check last comparison conflict
        if (compare_cycles[i].contains(first) or
compare_cycles[i].contains(second)) {
            // check whether a new cycle needs to be added
            if (i < cycles_count - 1) {
                compare_cycles[i + 1].insert(first);
                compare_cycles[i + 1].insert(second);
            } else {
                compare_cycles.push_back(std::set<size_t>{first, second});
            }
            return;
        }
    }
    // without any conflicts suspected -> add elements to first cycle
    compare_cycles[0].insert(first);
    compare_cycles[0].insert(second);
}

void myComparator(std::pair<long, long> &first, std::pair<long, long>
&second) {
    if (is_printing) {
        std::cout << first.first << " " << second.first << std::endl;
    }
    if (first.second > second.second) {
        std::swap(first.second, second.second);
    }
    comparator_count++;
    addPairToCycles(first.first, second.first);
}

void sortAndMergeTwoArrays(const std::shared_ptr<std::pair<long, long>[]>
&array,
                           long first_size,
                           long second_size,
                           long start = 0) {
    if (first_size <= 0 or second_size <= 0) {
        return;
```

```
    } else if (first_size == 1 and second_size == 1) {
        myComparator(array[start + 0], array[start + 1]);
        return;
    } else {
        // calculate index
        auto first_even_size = (long) round((double) first_size / 2);
        long first_odd_size = first_size - first_even_size;
        auto second_even_size = (long) round((double) second_size / 2);
        long second_odd_size = second_size - second_even_size;

        // create sub array and fill them
        auto even_prt = std::shared_ptr<std::pair<long, long>[]>(
                new std::pair<long, long>[first_even_size +
second_even_size]);

        for (long i = 0; i < first_even_size; i++) {
            even_prt[i] = array[start + 2 * i];
        }
        for (long i = 0; i < second_even_size; i++) {
            even_prt[first_even_size + i] = array[start + first_size + 2 *
i];
        }

        auto odd_prt = std::shared_ptr<std::pair<long, long>[]>(
                new std::pair<long, long>[first_odd_size + second_odd_size]);

        for (long i = 0; i < first_odd_size; i++) {
            odd_prt[i] = array[start + 2 * i + 1];
        }
        for (long i = 0; i < second_odd_size; i++) {
            odd_prt[first_odd_size + i] = array[start + first_size + 2 * i +
1];
        }

        // sort sub arrays
        sortAndMergeTwoArrays(even_prt, first_even_size, second_even_size);
        sortAndMergeTwoArrays(odd_prt, first_odd_size, second_odd_size);

        // merge sub arrays
        for (long i = 0; i < first_even_size; i++) {
            array[start + 2 * i] = even_prt[i];
        }
        for (long i = 0; i < second_even_size; i++) {
            array[start + first_size + 2 * i] = even_prt[first_even_size +
i];
        }
        for (long i = 0; i < first_odd_size; i++) {
            array[start + 2 * i + 1] = odd_prt[i];
        }
        for (long i = 0; i < second_odd_size; i++) {
            array[start + first_size + 2 * i + 1] = odd_prt[first_odd_size +
i];
        }

        // count comparators in merging
        for (long i = 1; i < first_size + second_size - 1; i += 2) {
            myComparator(array[start + i], array[start + i + 1]);
        }
    }
}

void batcherSort(const std::shared_ptr<std::pair<long, long>[]> &array_ptr,
long cur_arr_size, long start = 0) {
```

```
    // stop recursion for all small array
    if (cur_arr_size < 2) {
        return;
    }
    long mid_size = cur_arr_size / 2;

    batcherSort(array_ptr, mid_size, start);
    batcherSort(array_ptr, cur_arr_size - mid_size, start + mid_size);
    sortAndMergeTwoArrays(array_ptr, mid_size, cur_arr_size - mid_size,
start);
}

bool zeroOneTestBatcherSort() {
    is_printing = false;
    auto array_prt = std::shared_ptr<std::pair<long, long>[]>(new
std::pair<long, long>[MAX_TEST_ARRAY_SIZE]);

    for (long cur_array_size = 1, cur_max_value = 2;
         cur_array_size <= MAX_TEST_ARRAY_SIZE; cur_array_size++,
cur_max_value *= 2) {
        for (long i = 0; i < cur_max_value; i++) {
            auto cur_temp = i;
            for (long j = 0; j < cur_array_size; j++) {
                array_prt[j] = std::make_pair(j, cur_temp % 2);
                cur_temp /= 2;
            }
            batcherSort(array_prt, cur_array_size);
            for (long j = 1; j < cur_array_size; j++) {
                if (array_prt[j] < array_prt[j - 1]) {
                    return false;
                }
            }
        }
    }
    return true;
}

int main(int argc, char **argv) {
    if (argc != 2) {
        std::cout << "Program accept only one additional argument" <<
std::endl;
        return 0;
    }
    if (std::string(argv[1]) == "test") {
        std::cout << zeroOneTestBatcherSort();
        return 0;
    }
    long array_size = strtol(argv[1], nullptr, 10);
    if (array_size <= 0) {
        std::cout << "First argument must be positive" << std::endl;
        return 0;
    }
    std::cout << array_size << " 0 0" << std::endl;

    auto array_prt = std::shared_ptr<std::pair<long, long>[]>(new
std::pair<long, long>[array_size]);

    for (long i = 0; i < array_size; i++) {
        array_prt[i] = std::make_pair(i, i * (i + 23) % 1653);
    }

    batcherSort(array_prt, array_size);
    std::cout << comparator_count << std::endl << compare_cycles.size() <<
```

```
std::endl;
    return 0;
}
```