Московский государственный университет имени М.В. Ломоносова
Факультет вычислительной математики и кибернетики

Скрябин Глеб Денисович

Задание 1
Расписание сети сортировки

Отчет о работе по курсу "Параллельные высокопроизводительные вычисления"

Автор:
гр. 523

Москва, 12 ноября 2023

Содержание

1 Описание задания

1.1 Условие

Требуется 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.

1.2 Команда запуска

Параметр командной строки запуска: n. n >= 1 — количество элементов в упорядочиваемом массиве, элементы которого расположены на строках с номерами [0. . . n - 1]. Формат команды запуска: bsort n

1.3 Требования к работе программы

- Вывести в файл стандартного вывода расписание и его характеристики в представленном далее формате;
- Обеспечить возможность вычисления сети сортировки для числа элементов 1 <= n <= 10000;
- Предусмотреть полную проверку правильности сети сортировки для значений числа сортируемых элементов 1 <= n <= 24.

1.4 Формат файла результата

```
// Начало файла результата
n 0 0
cu_0  cd_0
cu_1  cd_1
...
cun_comp - 1  cdn_comp - 1
n_comp
n_tact
// Конец файла  результата
```

Listing 1: Формат файла результата

Здесь:

- n 0 0 — число сортируемых элементов, ноль, ноль. Да, вывести число элементов и два нуля;
- cu_i cd_i — номера строк, соединяемых i-м компаратором сравнения перестановки;
- n_comp — число компараторов;
- n_tact — число тактов сети сортировки.

2 Описание метода решения

2.1 Сети Бэтчера

Сети Бэтчера — наиболее быстродействующие из рассматриваемых масштабируемых сетей сортировки. Для построения сети обменной сортировки со слиянием можно использовать описываемый далее рекурсивный алгоритм или приводящий к аналогичному результату нерекурсивный алгоритм.

Для сортировки массива, содержащего p элементов с номерами [1, ..., p] следует разделить его на две части. В первой оставить $n = \lfloor \frac{p}{2} \rfloor$ элементов с номерами [1, ..., n], а во второй m = p – n элементов с номерами [n + 1, ..., p]. Далее следует отсортировать каждую из частей и объединить результаты сортировки с помощью (n, m)-сети нечетно-четного слияния Бэтчера. В сети нечетно-четного слияния отдельно объединяются элементы массивов с нечетными номерами и отдельно — с четными, после чего, с помощью заключительной группы компараторов, обрабатываются пары соседних элементов с номерами вида (2i, 2i + 1), где i — натуральные числа от 1 до $\lfloor \frac{p}{2} \rfloor - 1$.

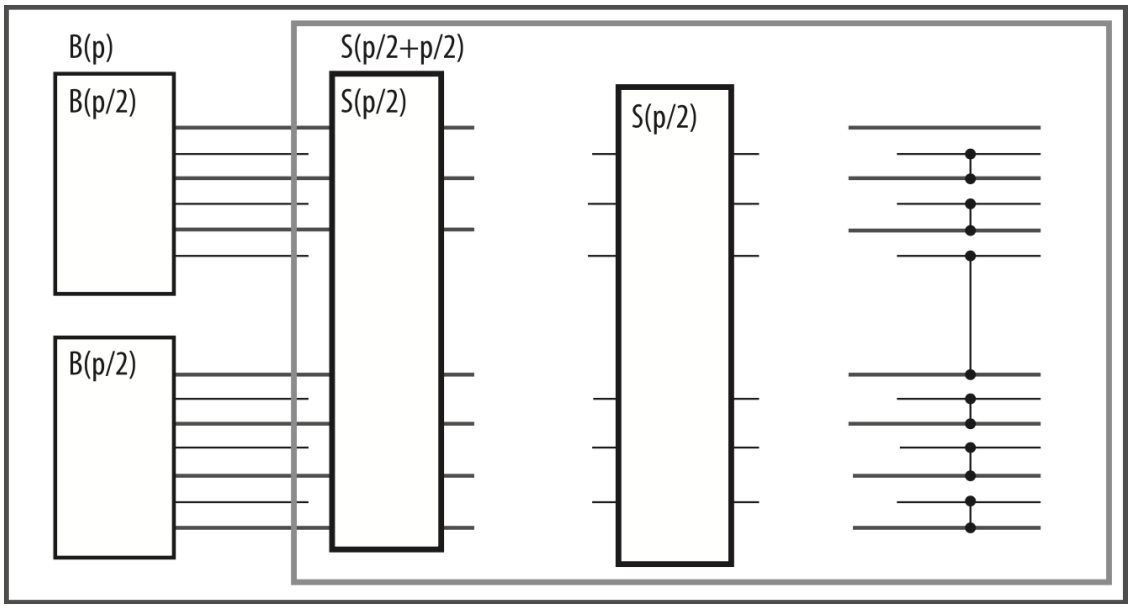

Рис. 1: Общая структура алгоритма построения сети обменной сортировки со слиянием Бэтчера

Общая структура алгоритма формирования сети сортировки массива длинны p приведена на рисунке 1. Для формирования сети важны только номера элементов (линий

данных), а не сами элементы. Группу линий удобно описывать с помощью тройки чисел вида (first, step, count). В такую группу будут входить линии с номерами first, first + step, ..., first + (count – 1) * step. Множество входящих в сеть компараторов формируется с помощью рекурсивных процедур **B** и **S**.

B(first, step, count) — процедура рекурсивного построения сети сортировки группы линий (first, step, count). Текст процедуры приведен на листинге 4.

S(first1, first2, step, count1, count2) — рекурсивная процедура слияния двух групп линий (first1, step, count1) и (first2, step, count2). Текст процедуры приведен на листингах 2 и 3.

2.2 Текст процедуры S(first1, first2, step, count1, count2)

```
unsigned _S (const size_t a,
             const size_t b,
             const size_t step,
             const size_t n,
             const size_t m) {
    if (n * m < 1) return 0;
    if (n == 1 && m == 1) {
        _addComparator(a, b);
        return 1;
    }

    size_t i;
    size_t n1 = n - n / 2; // количество нечетных строк в массиве a
    size_t m1 = m - m / 2; // количество   четных строк в массиве b

    // продолжение на след. странице
```

Listing 2: Текст процедуры S(first1, first2, step, count1, count2), часть 1/2

```
    // объединить нечетные линии
    unsigned tMerge1 = _S(a, b, 2 * step, n1, m1);

    // объединить четные линии
    unsigned tMerge2 = _S(a + step, b + step, 2 * step, n - n1, m - m1);

    // далее добавить цепочку компараторов, начиная со второй линии

    // компараторы между линиями первого массива
    for (i = 1; i < n - 1; i += 2) {
        _addComparator(a + step * i, a + step * (i + 1));
    }

    if (n % 2 == 0) {
        // компаратор между массивами
        _addComparator(a + step * (n - 1), b);
        i = 1;
    } else {
        i = 0;
    }

    // компараторы между линиями второго массива
    for(; i < m - 1; i += 2) {
        _addComparator(b + step * i, b + step * (i + 1));
    }

    return std::max(tMerge1, tMerge2) + 1;
}
```

Listing 3: Текст процедуры S(first1, first2, step, count1, count2), часть 2/2

2.3 Текст процедуры B(first, step, count)

```
unsigned _B (const size_t first,
             const size_t step,
             const size_t n) {
    if (n < 2) return 0;
    if (n == 2) {
        _addComparator(first, first + step);
        return 1;
    }

    // число элементов в первой половине массива
    size_t n1 = std::ceil(n / 2);

    // число элементов во второй половине массива
    size_t n2 = n - n1;

    // упорядочить первую половину массива
    unsigned t1 = _B(first, step, n1);

    // упорядочить вторую половину массива
    unsigned t2 = _B(first + step * n1, step, n2);

    // объединить упорядоченные части
    unsigned tMerge = _S(first, first + step * n1, step, n1, n2);

    return std::max(t1, t2) + tMerge;
}
```

Listing 4: Текст процедуры B(first, step, count)

3 Описание метода проверки

Для проведения проверки корректности работы программной реализации алгоритма было решено использовать сравнение результатов сортировки стандартной библиотеки std::sort и обменной сортировки со слиянием Бэтчера на всевозможных вариантах массивов, состоящих из 0 и 1, длинной N из диапазона значений [1...24].

Помимо этого был реализован дополнительный этап тестирования алгоритма по тому же принципу на массивах, состоящих их случайных элементов типа int32_t, длинной N = 10000.