Расписание сети сортировки

первое практическое задание по курсу

“Параллельные высокопроизводительные вычисления”

Михальцов Данила Алексеевич

528 группа (1 курс магистратуры)

12 ноября 2023

Описание условия

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.

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

cu_0 cd_0

cu_1 cd_1

…

cu_{n_comp-1} cd_{n_comp-1}

n_comp

n_tact

Конец файла результата

Здесь:

n 0 0 – число сортируемых элементов, ноль, ноль, cu_i cd_i *– номера строк, соединяемых i-м компаратором сравнения перестановки, n_comp – число компараторов, n_tact – число тактов сети сортировки.*

Описание метода решения

Детали реализации

Программа написана на языке C++ с использованием функций только стандартной библиотеки. Для комиляции использовался компилятор g++. Команда для компиляции программы выглядит так:

```
g++ bsort.cpp -o bsort
```

Для запуска построения расписания сети сортировки требуется передать количество элементов сортируемого массива в качестве аргумента командной строки:

```
./bsort 6
```

Для запуска проверки решения на всех наборах из нулей и единиц длиной 1 <= n <= 24 требуется передать в качестве аргумента командной строки число -1:

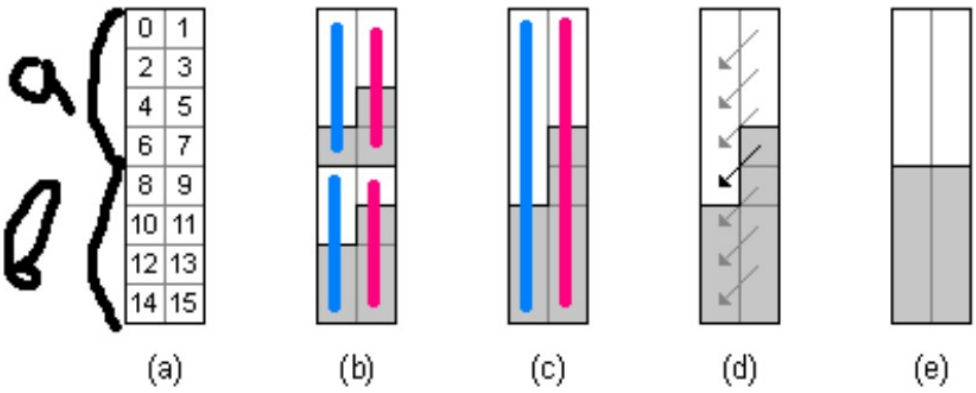
```
./bsort -1
```

Детали решения

Для получения сети сортировки использовалась чётно-нечётная сортировка Бэтчера. Реализованы две функции: `odd_even_merge` и `odd_even_sort`. Далее описана работа каждой из них.

`odd_even_merge(int a_start, int b_start, int step, int n_a, int n_b, Comparators_vector &c)`

Получая на вход два отсортированных массива, функция сортирует исходный массив (подготавливает компараторы для этого). Поскольку оба массива отсортированы, то и их части, выделенные на рисунке под пунктом “b” также отсортированы. Тогда рекурсивно вызвав на них ту же функцию, мы получим два отсортированных массива (пункт “c”). Для их слияния достаточно показанных компараторов, так как в силу отсортированности изначальных массивов a и b в правом столбце может быть не больше двух бóльших элементов. Таким образом, мы получили отсортированный массив.



`odd_even_sort(int first, int step, int n, Comparators_vector &c)`

Выполняет сортировку массива (подготавливает компараторы для этого). Сначала она делит массив на две половины, затем сортирует каждую из них этой же функцией, а затем проводит слияние функцией `odd_even_merge`. В тривиальных случаях поведение отличается: при n==1 массив уже отсортирован, а следовательно, ничего делать не нужно, а при n==2 для сортировки требуется произвести ровно одно сравнение между этими двумя элементами.

Подсчёт тактов

Подсчёт количества тактов реализован следующим образом: выполняется проход по компараторам и производится подсчёт минимального необходимого количества тактов для выполнения обменов i-того процессора следующим образом: если требуется совершить обмен информацией между i-тым и j-тым процессором, то для этого потребуется max(текущий ответ для i-того процессора + 1, текущий ответ для j-того процессора + 1). Это значение записывается как минимальное необходимое число тактов для обоих этих процессоров. Ответ – максимум из этих значений для всех процессоров после прохода по всем компараторам.

Описание метода проверки

Для проверки корректности сортировки была проведена проверка корректности сортировки всех наборов из 0 и 1 длины 1 <= n <= 24. Все проверки прошли успешно.

Для генерации таких последовательностей использовалась следующая логика: для того, чтобы перебрать все наборы нулей и единиц длины n, достаточно взять набор (0, …, 0) и прибавлять 1 к этому набору как к двоичному числу, перенося 1 в следующий разряд при переполнении разрядов. Остановить перебор следует, когда нужно будет сделать перенос в несуществующий разряд, т.е. прибавить 1 к двоичному числу, соответствующему набору (1, …, 1).

К каждому сгенерированному набору из нулей и единиц применялось сгенерированное расписание сети сортировки и далее проверялась монотонность полученного двоичного набора. Если набор отсортирован некорректно, выбрасывается исключение, приводящее к аварийному завершению программы с соответствующим комментарием.

Приложение 1: исходный текст программы

Код приложен отдельным файлом.