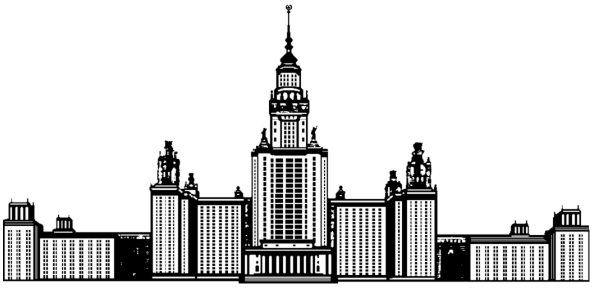

Московский государственный университет имени М.В. Ломоносова
Факультет вычислительной математики и кибернетики
Кафедра вычислительных методов

Параллельные высокопроизводительные вычисления

# Задание 1. Расписание сети слияния

**Выполнила:**
студентка 504 группы
Погосбекян Мария Михайловна

**Дата подачи:**
01.11.2022

Москва, 2022

# 1 Описание условия

Разработать последовательную программу вычисления расписания сети объединения двух упорядоченных массивов с размерами $p_1$ и $p_2$, числа использованных компараторов, числа тактов, необходимых для её выполнения на $p = p_1 + p_2$ процессорах.

Число тактов сортировки при параллельной обработке не должно превышать числа тактов, затрачиваемых соответствующей сетью четно-нечетного слияния Бетчера.

Параметры командной строки запуска: $p_1$, $p_2$.

$p_1 \geq 1$, $p_2 \geq 0$ – числа элементов в упорядоченных объединяемых массивах, элементы которых расположены в строках с номерами $[0 \ldots p_1 - 1]$ и $[p_1 \ldots p_1 + p_2 - 1]$ соответственно.

**Формат команды запуска:** Bjoin p1 p2

Требуется:

- вывести в файл стандартного вывода расписание и его характеристики в представленном далее формате
- обеспечить возможность вычисления сети сортировки для числа элементов $1 <= p_1 + p_2 <= 10000$
- предусмотреть полную проверку правильности сети сортировки для значений числа сортируемых элементов $1 \leq p_1 + p_2 \leq 24$

**Формат файла результата:**

**Начало файла результата**

$p_1$ $p_2$ 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:

$p_1$, $p_2$, 0 – размеры первого и второго упорядоченных массивов, число 0

$cu_i$ $cd_i$ – номера строк, соединяемых i-м компаратором сравнения - перестановки

$n_{comp}$ – общее число компараторов

$n_{tact}$ – общее число тактов сети слияния

# 2 Описание метода решения

Сети Бэтчера – наиболее быстродействующие из масштабируемых сетей сортировки. Для построения сети обменной сортировки со слиянием используется следующий рекурсивный алгоритм. Чтобы отсортировать массив, состоящий из $p$ элементов с номерами $[1, \ldots, p]$, нужно поделить его на две части: в первой будет $n = \lceil \frac{p}{2} \rceil$ элементов с номерами $[1, \ldots, n]$ , во второй – $m = p - n$ элементов с номерами $[n + 1, \ldots, p]$. Далее с помощью функции $Sort$ сортируется каждая из частей, а затем с помощью функции $Join$ происходит объединение отсортированных частей.

Функция $Sort$ – рекурсивное построение сети сортировки группы линий. Делит массив на две части. Одна состоит из $n$ элементов, другая из $m$, дальше функция $Sort$ вызывается для каждой части массива, а затем вызывает функцию $Join$ для объединения упорядоченных частей.

Так как по условию задания на вход поступает 2 упорядоченных массива, то функция $Sort$ не нужна.

Функция $Join$ – рекурсивное слияние двух групп линий. В сети нечетно-четного слияния отдельно объединяются массивов с нечетными номерами и отдельно – с четными. Далее с помощью заключительной группы компараторов обрабатываются пары соседних элементов с номерами вида $(2i, 2i + 1)$, где $i$ – натуральные числа от 1 до $\lfloor \frac{p}{2} \rfloor - 1$.

# 3 Описание метода проверки

Для проверки будем использовать принцип нулей и единиц: если сеть с $p$ входами сортирует в порядке неубывания все $2^p$ последовательности из 0 и 1, то она будет сортировать в том же порядке любую последовательность чисел. Будем генерировать различные последовательности из 0 и 1 длиной от 1 до 24. Если 2 последовательности упорядочены, то запускаетя функция $Join$, далее где нужно меняются местами элементы, затем проверяется, отстортирован ли итоговый массив, если да, то печатается "$Good\ job,\ bro$ :)!", в противном случае – "$Oops,\ tests\ failed: c$".

# A Приложение

## Код программы

```
   #include <iostream>
#include <vector>
#include <string>
#include <cmath>
#include <bitset>
#include <algorithm>

using namespace std;

vector<pair<int, int>> vComparators;

void Join(int iFirst1, int iFirst2, int iStep, int iCount1, int iCount2) {
    int iCountOdd1, iCountEven2, i;

    if (iCount1 * iCount2 < 1) return;
    if (iCount1 == 1 && iCount2 == 1) {
        vComparators.push_back(make_pair(iFirst1, iFirst2));
        return;
    }

    iCountOdd1 = iCount1 - (iCount1 / 2);
    iCountEven2 = iCount2 - (iCount2 / 2);

    Join(iFirst1, iFirst2, 2 * iStep, iCountOdd1, iCountEven2);
    Join(iFirst1 + iStep, iFirst2 + iStep, 2 * iStep, iCount1 - iCountOdd1,
   iCount2 - iCountEven2);

    for (i = 1; i < iCount1 - 1; i += 2) {
        vComparators.push_back(make_pair(iFirst1 + iStep * i, iFirst1 + iStep *
    (i + 1)));
    }
    if (iCount1 % 2 == 0) {
        vComparators.push_back(make_pair(iFirst1 + iStep * (iCount1 - 1),
   iFirst2));
        i = 1;
    }
    else i = 0;

    for (; i < iCount2 - 1; i += 2) {
        vComparators.push_back(make_pair(iFirst2 + iStep * i, iFirst2 + iStep *
    (i + 1)));
    }

}

bool bOK = true;
void check() {
    for (int iSize = 1; iSize <= 24; iSize++) {
        for (int i = 0; i <= pow(2, iSize) - 1; i++) {
            string curSeq = bitset<24>(i).to_string();
            curSeq = curSeq.substr(24 - iSize);

            for (int p1 = 1; p1 <= iSize; p1++) {
                int p2 = iSize - p1;
                bool bOKFirst = is_sorted(begin(curSeq), begin(curSeq) + p1 -
   1);
                bool bOKSecond = is_sorted(begin(curSeq) + p1, end(curSeq));

                if (bOKFirst && bOKSecond) {
```

```
55                      vComparators.clear();
56
57                      Join(0, p1, 1, p1, p2);
58
59                      for (int j = 0; j < vComparators.size(); j++) {
60                          if (((int)curSeq[vComparators[j].first]) > ((int)curSeq
    [vComparators[j].second]))
61                              swap(curSeq[vComparators[j].first], curSeq[
    vComparators[j].second]);
62                      }
63                      if (!is_sorted(begin(curSeq), end(curSeq))) bOK = false;
64                      //cout << curSeq << " p1 = " << p1 << " p2 = " << p2 <<
    endl;
65                  }
66              }
67          }
68      }
69      if (bOK) cout << "Good job, bro:)!" << endl;
70      else cout << "Oops, tests failed:c" << endl;
71  }
72
73  int CountTackts(vector<pair<int, int>> vCur, int iSize) {
74      int iMax1, iMax2;
75      vector<int>vTackts(iSize);
76
77      for (int i = 0; i < vCur.size(); i++) {
78          iMax1 = max(vTackts[vComparators[i].first], vTackts[vComparators[i].
    second]);
79          vTackts[vComparators[i].first] = iMax1 + 1;
80          vTackts[vComparators[i].second] = iMax1 + 1;
81      }
82
83      iMax2 = vTackts[0];
84      for (int i = 1; i < iSize; i++) {
85          if (vTackts[i] > iMax2) iMax2 = vTackts[i];
86      }
87
88      return iMax2;
89   };
90
91
92  int main(int argc, char** argv)
93  {
94      int p1, p2;
95      vector<int> v1, v2;
96
97      cout << argv[1] << " " <<  argv[2] << " " << "0" << endl;
98      p1 = stoi(argv[1]);
99      p2 = stoi(argv[2]);
100
101     Join(0, p1, 1, p1, p2);
102
103     for (int i = 0; i < vComparators.size(); i++) {
104         cout << vComparators[i].first << " " << vComparators[i].second << endl;
105     }
106     cout << vComparators.size() << endl;
107     cout << CountTackts(vComparators, p1 + p2) << endl;
108     //check();
109
110
111     return 0;
112 }
```