Санкт-Петербургский политехнический университет Петра Великого

Институт компьютерных наук и технологий

**Высшая школа искусственного интеллекта**

**Лабораторная работа 5**

Параллельные алгоритмы сортировки

по дисциплине «Параллельное программирование»

Выполнил

студент гр.3530203/80102 Л.Д. Челищева

<*подпись*>

Руководитель К.А. Туральчук

<*подпись*>

«\_\_\_» \_\_\_\_\_\_\_\_\_\_ 2021 г.

Санкт-Петербург

2021

# Постановка задачи

**Вариант 6: Быстрая блочная сортировка (нерекурсивная)**

Задачи:

1. Реализовать последовательный алгоритм.
2. Реализовать параллельный алгоритм. Для локальной сортировки элементов блоков использовать библиотечный алгоритм быстрой сортировки Array.Sort.
3. Выполнить анализ эффективности при разном объеме данных и при разной упорядоченности данных (неупорядоченный набор, упорядоченный по возрастанию, упорядоченный по убыванию). Построить графики зависимости ускорения алгоритма от объема данных.

## Описание алгоритма:

Элементы разбиваются на блоки

Число блоков:Изображение выглядит как текст, часы, датчик

Автоматически созданное описание

Число потоков/процессоров:Изображение выглядит как текст, часы, датчик

Автоматически созданное описание

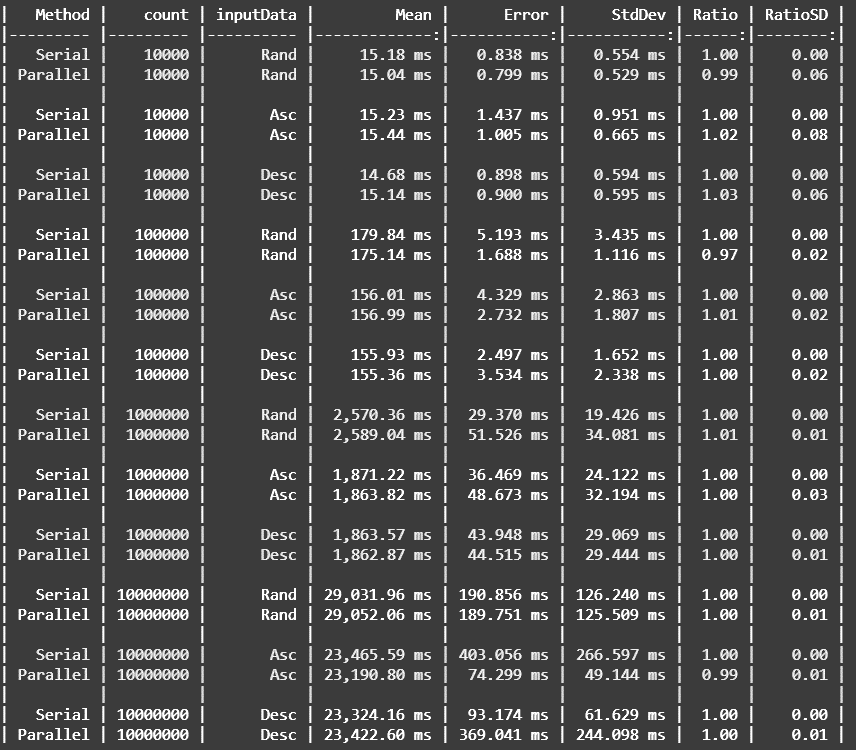
Выполняется N итераций взаимодействия пар блоков. На каждой i-итерации взаимодействуют блоки, номера которых различаются только в (N-i)-разряде в битовом представлении.

На каждой i-итерации выбирается  ведущих элемента, по одному элементу для всех пар, номера блоков которых имеют одинаковые биты в разрядах [N; N-i);

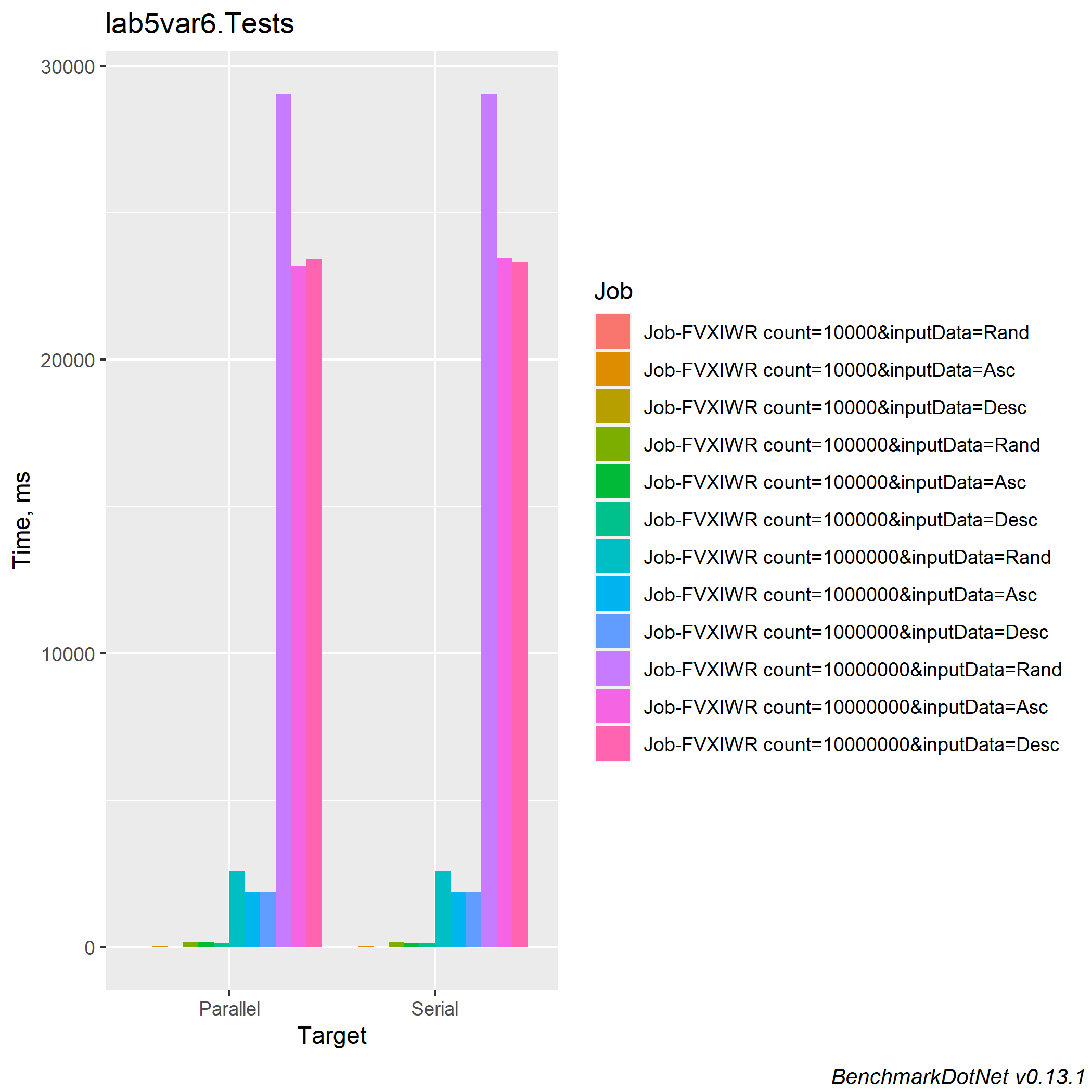
При взаимодействии блок с меньшим номером получает все элементы, меньшие ведущего элемента; блок с большим номером получает все элементы, большие ведущего элемента.

Задача распараллеливания выполнялась на этапе перестановки элементов в блоках – с каждой парой блоков работает отдельный поток.

# Результаты тестирования



Тестирование показало результаты, отличные от ожидаемых – параллельный алгоритм не показал никаких преимуществ.



# Полный код программы

using System;

using System.Collections.Generic;

using System.Threading;

using BenchmarkDotNet.Attributes;

using BenchmarkDotNet.Engines;

using BenchmarkDotNet.Running;

namespace lab5var6

{

[RPlotExporter]

[SimpleJob(RunStrategy.Monitoring, targetCount: 10, invocationCount: 1)]

public class Tests

{

[Params(10000, 100000, 1000000, 10000000)]

public int count; // Число элементов в массиве

public int[] arr; // Массив чисел

public enum InputType { Rand, Asc, Desc } //Виды наборов чисел

[Params(InputType.Rand, InputType.Asc, InputType.Desc)]

public InputType inputData;

private Serial serial = new Serial();

private Parallel parallel = new Parallel();

[IterationSetup]

public void Setup()

{

arr = new int[count];

//Случайная последовательность чисел

if (inputData == InputType.Rand)

RandArr();

//Числа по возрастанию

if (inputData == InputType.Asc)

InitAsc();

//Числа по убыванию

if (inputData == InputType.Desc)

InitDesc();

}

private void RandArr()

{

Random rnd = new Random();

for (int i = 0; i < count; i++)

{

arr[i] = rnd.Next(1, 10000);

}

}

private void InitAsc()

{

for (int i = 0; i < count; i++)

{

arr[i] = i;

}

}

private void InitDesc()

{

for (int i = 0; i < count; i++)

{

arr[i] = count - i;

}

}

[Benchmark(Baseline = true)]

public void Serial()

{

for (int i = 0; i < 100; i++) serial.QBSort(ref arr);

}

[Benchmark]

public void Parallel()

{

for (int i = 0; i < 100; i++) parallel.QBSort(ref arr);

}

}

public class Serial

{

public static int N = 4;

public static int q = (int)Math.Pow(2, N); //Число блоков

public static List<int>[] blocks = new List<int>[q]; //Массив блоков;

private static int countOld = q;

private static int countNew = countOld / 2;

public void Change(object o)//Переместить элементы 2 блоков по ведущему элементу

{

int[] tmp = (int[])o;

int bl1 = tmp[0];

int bl2 = tmp[1];

int pivot = tmp[2];

if (blocks[bl1].Count == 0 && blocks[bl2].Count == 0) return;

List<int> temp = new List<int>();

for (int i = 0; i < blocks[bl1].Count; i++) temp.Add(blocks[bl1][i]);

for (int i = 0; i < blocks[bl2].Count; i++) temp.Add(blocks[bl2][i]);

blocks[bl1].Clear();

blocks[bl2].Clear();

for (int i = 0; i < temp.Count; i++)

{

if (temp[i] > pivot)

{

blocks[bl2].Add(temp[i]);

}

else

{

blocks[bl1].Add(temp[i]);

}

}

}

public void QBSort(ref int[] arr)

{

//Разделить блоки

int countBlocks = arr.Length / q;

for (int i = 0; i < q - 1; i++)

{

blocks[i] = new List<int>();

for (int j = i \* countBlocks; j < (i + 1) \* countBlocks; j++)

{

blocks[i].Add(arr[j]);

}

}

blocks[q - 1] = new List<int>();

for (int j = countBlocks \* (q - 1); j < arr.Length; j++)

{

blocks[q - 1].Add(arr[j]);

}

//Работа сортировки

while (countNew > 0)

{

for (int i = 0; i < q; i += countOld)

{

int pivot = 0;

for (int j = i; j < i + countNew; j++)

{

if (blocks[j].Count != 0)

{

pivot = blocks[j][blocks[j].Count - 1];

break;

}

}

for (int j = i; j < i + countNew; j++)

{

Change(new int[] { j, j + countNew, pivot });

}

}

countOld = countNew;

countNew /= 2;

}

for (int i = 0; i < blocks.Length; i++)

{

blocks[i].Sort();

}

//Склеить блоки в один массив

int ind = 0;

for (int i = 0; i < blocks.Length; i++)

{

for (int j = 0; j < blocks[i].Count; j++)

{

arr[ind] = blocks[i][j];

ind++;

}

}

}

}

public class Parallel

{

public static int N = 4;

public static int q = (int)Math.Pow(2, N); // число блоков

public static List<int>[] blocks = new List<int>[q]; // массив блоков

private static int countOld = q;

private static int countNew = countOld / 2;

public static int p = q / 2; // число потоков/процессоров

public static Thread[] threads = new Thread[p];

public void Change(object o)

{

int[] tmp = (int[])o;

int bl1 = tmp[0];

int bl2 = tmp[1];

int pivot = tmp[2];

if (blocks[bl1].Count == 0 && blocks[bl2].Count == 0) return;

List<int> temp = new List<int>();

for (int i = 0; i < blocks[bl1].Count; i++) temp.Add(blocks[bl1][i]);

for (int i = 0; i < blocks[bl2].Count; i++) temp.Add(blocks[bl2][i]);

blocks[bl1].Clear();

blocks[bl2].Clear();

for (int i = 0; i < temp.Count; i++)

{

if (temp[i] > pivot)

{

blocks[bl2].Add(temp[i]);

}

else

{

blocks[bl1].Add(temp[i]);

}

}

}

public void QBSort(ref int[] arr)

{

//Разделить блоки

int countBlocks = arr.Length / q;

for (int i = 0; i < q - 1; i++)

{

blocks[i] = new List<int>();

for (int j = i \* countBlocks; j < (i + 1) \* countBlocks; j++)

{

blocks[i].Add(arr[j]);

}

}

blocks[q - 1] = new List<int>();

for (int j = countBlocks \* (q - 1); j < arr.Length; j++)

{

blocks[q - 1].Add(arr[j]);

}

//Работа сортировки

while (countNew > 0)

{

int tr = 0;

for (int i = 0; i < q; i += countOld)

{

int pivot = 0;

for (int j = i; j < i + countNew; j++)

{

if (blocks[j].Count != 0)

{

pivot = blocks[j][blocks[j].Count - 1];

break;

}

}

//Распараллеливание попарного сравнения блоков

for (int j = i; j < i + countNew; j++)

{

threads[tr] = new Thread(Change);

threads[tr].Start(new int[] { j, j + countNew, pivot });

tr++;

}

}

for (int i = 0; i < p; i++)

{

threads[i].Join();

}

countOld = countNew;

countNew /= 2;

}

for (int i = 0; i < blocks.Length; i++)

{

blocks[i].Sort();

}

//Склеить блоки в один массив

int ind = 0;

for (int i = 0; i < blocks.Length; i++)

{

for (int j = 0; j < blocks[i].Count; j++)

{

arr[ind] = blocks[i][j];

ind++;

}

}

}

}

class Program

{

static void Main(string[] args)

{

BenchmarkRunner.Run<Tests>();

Console.ReadKey();

}

}

}

# Выводы

В ходе лабораторной работы был рассмотрен алгоритм быстрой блочной сортировки (нерекурсивная версия), который был выполнен в двух версиях – последовательной и параллельной. При этом считывание и анализ результатов тестирования был выполнен с помощью пакета BenchmarkDotNet.

Согласно результатам тестирования, подобная параллельная реализация не дает преимущества по времени перед последовательной.