Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

Отчет по лабораторной работе №4 по курсу "Анализ алгоритмов"

Тема Параллельное программирование

Студент Козлова И.В.

Группа ИУ7-52Б

Оценка (баллы)

Преподаватель Волкова Л.Л.

Москва — 2021 г.

Оглавление

Введение

Многопоточность — способность центрального процессора (CPU) или одного ядра в многоядерном процессоре одновременно выполнять несколько процессов или потоков, соответствующим образом поддерживаемых операционной системой. Этот подход отличается от многопроцессорности, так как многопоточность процессов и потоков совместно использует ресурсы одного или нескольких ядер: вычислительных блоков, кэш-памяти ЦПУ или буфера перевода с преобразованием (TLB).

В тех случаях, когда многопроцессорные системы включают в себя несколько полных блоков обработки, многопоточность направлена на максимизацию использования ресурсов одного ядра, используя параллелизм на уровне потоков, а также на уровне инструкций. Поскольку эти два метода являются взаимодополняющими, их иногда объединяют в системах с несколькими многопоточными ЦП и в ЦП с несколькими многопоточными ядрами.

Многопоточная парадигма стала более популярной с конца 1990-х годов, поскольку усилия по дальнейшему использованию параллелизма на уровне инструкций застопорились. Смысл многопоточности — квазимногозадачность на уровне одного исполняемого процесса. Значит, все потоки процесса помимо общего адресного пространства имеют и общие дескрипторы файлов. Выполняющийся процесс имеет как минимум один (главный) поток.

Многопоточность (как доктрину программирования) не следует путать ни с многозадачностью, ни с многопроцессорностью, несмотря на то, что операционные системы, реализующие многозадачность, как правило, реализуют и многопоточность.

Достоинства:

- облегчение программы посредством использования общего адресного пространства;
- меньшие затраты на создание потока в сравнении с процессами;
- повышение производительности процесса за счёт распараллеливания процессорных вычислений;

- если поток часто теряет кэш, другие потоки могут продолжать использовать неиспользованные вычислительные ресурсы.

Недостатки:

- несколько потоков могут вмешиваться друг в друга при совместном использовании аппаратных ресурсов [1];
- с программной точки зрения аппаратная поддержка многопоточности более трудоемка для программного обеспечения [2];
- проблема планирования потоков;
- специфика использования.

Вручную настроенные программы на ассемблере, использующие расширения MMX или AltiVec и выполняющие предварительные выборки данных, не страдают от потерь кэша или неиспользуемых вычислительных ресурсов. Таким образом, такие программы не выигрывают от аппаратной многопоточности и действительно могут видеть ухудшенную производительность из-за конкуренции за общие ресурсы.

Однако несмотря на количество недостатков, перечисленных выше, многопоточная парадигма имеет большой потенциал на сегодняшний день и при должном написании кода позволяет значительно ускорить однопоточные алгоритмы.

В рамках выполнения работы необходимо решить следующие задачи.

1. Изучения основ параллельных вычислений
2. Применение изученных основ для реализации многопоточной сортировки.
3. Получения практических навыков.
4. Произвести сравнительный анализ параллельной и однопоточной реализации алгоритма сортировки.
5. Экспериментальное подтверждение различий во временной эффективности реализации однопоточной и многопоточной ранговой сортировки.

6. Выбор и обоснование языка программирования, для решения данной задачи.

7. Описание и обоснование полученных результатов в отчете о выполненной лабораторной работе, выполненного как расчётно-пояснительная записка к работе.

1 Аналитическая часть

В данной лабораторной работе предполагается распараллеливание знакомого нам алгоритма. В качестве такого алгоритма был выбран алгоритм ранговой сортировки. В данной части будет рассмотрено тоеритическое описание алгоритма.

1.1 Сортировка

Сортировка - это процесс перегруппировки заданной последовательности (кортежа) объектов в некотором определеннном порядке. Такой определенный порядок позволяет, в некоторых случаях, эффективнее и удобнее работать с заданной последовательностью. В частности, одной из целей сортировки является облегчение задачи поиска элемента в отсортированном множестве.

Существует множество различных методов сортировки данных. Однако любой алгоритм сортировки можно разбить на три основные части:

- сравнение, определяющее упорядочность пары элементов;
- перестановка, меняющая местами пару элементов;
- собственно сортирующий алгоритм, который осуществляет сравнение и перестановку элементов данных до тех пор, пока все эти элементы не будут упорядочены.

Одной из важнейшей характеристикой любого алгоритма сортировки является скорость его работы, которая определяется функциональной зависимостью среднего времени сортировки последовательностей элементов данных, определенной длины, от этой длины.

1.2 Ранговая сортировка последовательности

Алгоритмы ранговой сортировки [3] основаны на вычислении рангов элементов и их упорядочении в соответствии с полученным рангом. Ранги элементов могут быть определены любым способом, однако в практике зачастую достаточным является следующая формула вычисления ранга:

$$rank(s_i) = \sum rank(R(s_i, s_j)), \tag{1.1}$$

где ранжируются результаты ($R(s_i, s_j)$ операций попарно сравнения элементов и выполняется суммирование полученных рангов.

Алгоритм можно выполнить следующими шагами для каждого элемента массива.

1. Подсчет количества чисел, меньше, чем исследуемое (в случае, если числа равны, то сравнивается индекс).

2. В результирующий массив в ячейку с индексом, равным количеству числе, меньших, чем исследуемое (подсчитано в шаге 1) поставить исследуемое число.

3. Повторить шаги 1-2 для всех элементов массива.

Пример по шагам представлен на рисунке 1.1.

Массив A	5	8	-1	0	4	3
Числа, которые меньше 5	5	8	-1	0	4	3
Массив B после 1ой итерации	0	0	0	0	5	0
Числа, которые меньше 8	5	8	-1	0	4	3
Массив B после 2ой итерации	0	0	0	0	5	8
Числа, которые меньше -1	5	8	-1	0	4	3
Массив B после 3ой итерации	-1	0	0	0	5	8
Числа, которые меньше 0	5	8	-1	0	4	3
Массив B после 4ой итерации	-1	0	0	0	5	8
Числа, которые меньше 4	5	8	-1	0	4	3
Массив B после 5ой итерации	-1	0	0	4	5	8
Числа, которые меньше 3	5	8	-1	0	4	3
Массив B после 6ой итерации	-1	0	3	4	5	8

Рисунок 1.1 – Пример сортировки

Вывод

В данном разделе были рассмотрены основополагающие материалы, которые в дальнейшем потребуются при параллельной и однопоточной реализации алгоритма ранговой сортировки.

2 Конструкторская часть

В данном разделе будет рассмотрены схемы исследуемого алгоритма.

2.1 Схема алгоритма

На рисунке 2.1 представлена схема алгоритма ранговой сортировки.

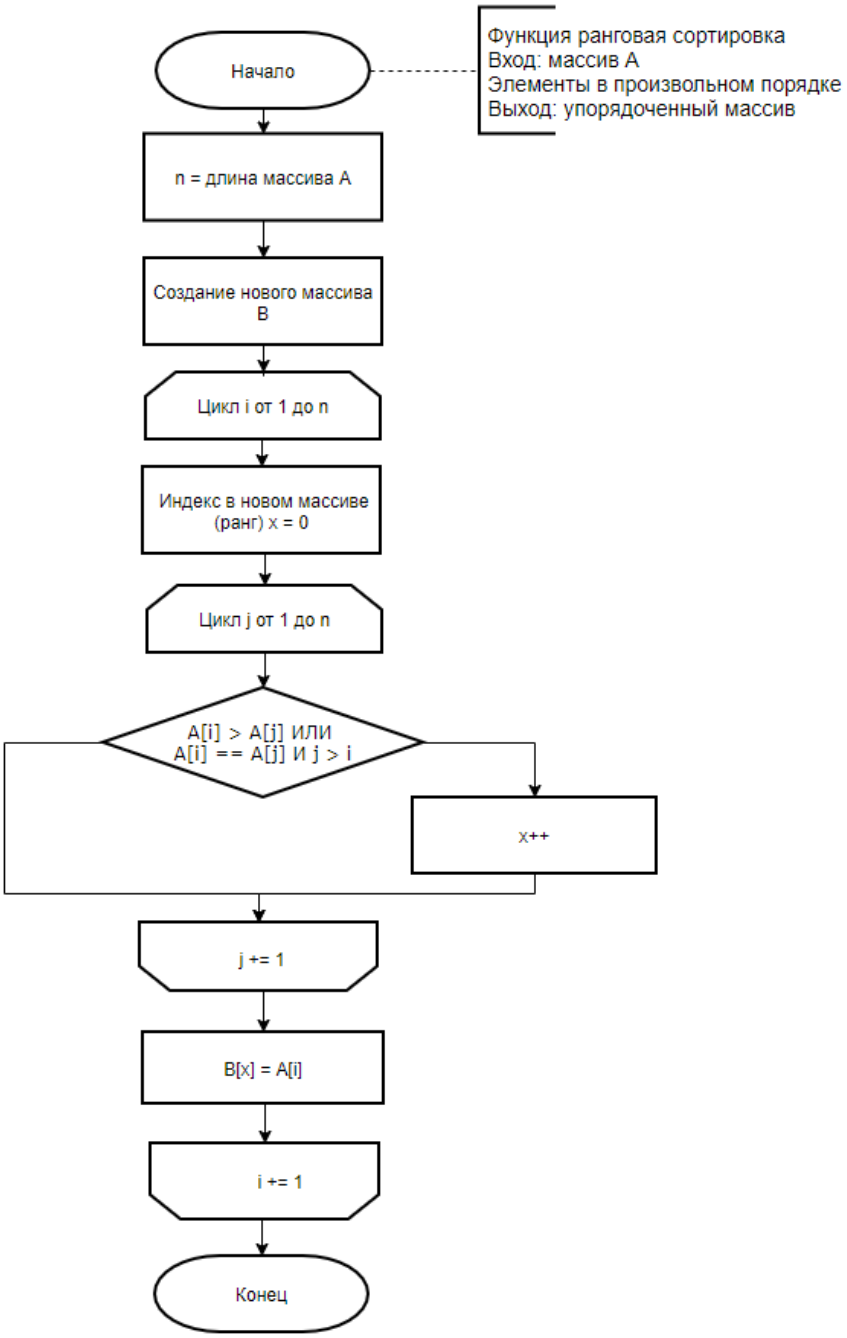

Рисунок 2.1 – Схема алгоритма ранговой сортировки

Вывод

В данном разделе были рассмотрены схемы однопоточной и многопоточной реализации алгоритма ранговой сортировки.

3 Технологическая часть

В данном разделе будут приведены требования к программному обеспечению, средства реализации и листинги кода.

3.1 Выбор языка программирования

В данной лабораторной работе использовался язык программирования - C# [4].

Данный язык является нативным.

В качестве среды разработки выбор сделан в сторону Visual Studio 2019 [5].

3.2 Сведения о модулях программы

Программа состоит из двух модулей:

1. Program.cs - главный файл программы, в котором содержится точка входа в программу.

2. Sort.cs - файл с реализацией алгоритма сортировки в двух варинатах (обычный и многопоточный).

3.3 Листинг кода

В листингах 3.1, 3.2 представлены реализации алгоритма ранговой сортировки многопоточной и обычной.

Также на листинге 3.3 представлен вариант реализации многопоточной программы для ранговой сортировки для замеров на больших массивов при 4 потоках.

Листинг 3.1 – Алгоритм ранговой сортировки (обычный)

```
1 public static string Sequential(ref int[] a)
2 {
```

```
3       int n = a.Length;
4       int[] b = new int[n];
5       int x;
6
7       for (int i = 0; i < n; i++)
8       {
9           x = 0;
10          for (int j = 0; j < n; j++)
11              if (a[i] > a[j] || (a[i] == a[j] && j > i))
12                  x++;
13          b[x] = a[i];
14      }
15      return str;
16  }
```

Листинг 3.2 – Алгоритм ранговой сортировки (многопоточный)

```
1   public static string Parallel(int[] a)
2   {
3       int M = 4;
4       Thread[] thrs = new Thread[M];
5       int n = a.Length;
6
7       for (int i = 0; i < M; i++)
8       {
9           int thr = i;
10          thrs[i] = new Thread(() =>
11          {
12              for (int j = thr; j < n; j += M)
13              {
14                  int x = 0;
15                  for (int k = 0; k < n; k++)
16                      if (a[j] > a[k] || (a[j] == a[k] && k > j))
17                          x++;
18                  b[x] = a[j];
19              }
20          });
21          thrs[i].Start();
22      }
23      for (int i = 0; i < M; i++)
24          thrs[i].Join();
25      return str;
```

```
26 }
```

Листинг 3.3 – Алгоритм ранговой сортировки (обычный)

```
1  public static int[] SortParall(this int[] a)
2  {
3      int n = a.Length;
4      int[] b = new int[n];
5      Parallel.ForEach(Partitioner.Create(0, n), range =>
6      {
7          int x;
8          for (int i = range.Item1; i < range.Item2; i++)
9          {
10             x = 0;
11
12             for (int j = 0; j < n; j++)
13                 if (a[i] > a[j] || (a[i] == a[j] && j > i))
14                     x++;
15             b[x] = a[i];
16         }
17     });
18     return b;
19 }
```

3.4 Тестирование

В таблице 3.1 приведены тесты для функций, реализующих алгоритмы сортировки. Тесты пройдены успешно.

Таблица 3.1 – Функциональные тесты

Входной массив	Ожидаемый результат	Результат
$[1, 2, 3, 4]$	$[1, 2, 3, 4]$	$[1, 2, 3, 4]$
$[5, 4, 3, 2, 1]$	$[1, 2, 3, 4, 5]$	$[1, 2, 3, 4, 5]$
$[3, 2, -5, 0, 1]$	$[-5, 0, 0, 2, 3]$	$[-5, 0, 0, 2, 3]$
$[4]$	$[4]$	$[4]$
$[]$	$[]$	$[]$

Вывод

В данном разделе были разобраны листинги показывающие работу как однопоточного, так и многопоточного алгоритма ранговой сортировки.

4 Исследовательская часть

В данном разделе будет произведено сравнение вышеизложенных алгоритмов.

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялось тестирование, следующие.

- Операционная система: Windows 10 [6] x86_64.
- Память: 8 GiB.
- Процессор: 11th Gen Intel® Core™ i5-1135G7 @ 2.40GHz [7].
- 4 физических ядра и 8 логических ядра.

Тестирование проводилось на ноутбуке, включенном в сеть электропитания. Во время тестирования ноутбук был нагружен только встроенными приложениями окружения, а также непосредственно системой тестирования.

4.2 Время выполнения алгоритмов

Результаты замеров приведены в таблицах 4.1 и 4.2. На рисунках 4.1, 4.2 и 4.3 приведены графики зависимостей времени работы реализации алгоритма ранговой сортировки от размеров массивов для различного количества потоков.

Таблица 4.1 – Время (мс) работы реализации алгоритма ранговой сортировки

Кол-во потоков	**Размер массива**		
	1000	**10000**	**500000**
1	237.51	661.43	14021.29
4	132.10	567.41	13848.28
8	38.80	445.27	12263.14
16	26.69	423.90	8434.71
32	12.48	493.91	7054.88

Таблица 4.2 – Время (мс) работы реализации алгоритма ранговой сортировки при константном количестве потоков (4 потока)

Размер	**Тип алгоритма**	
	Обычный	**Многопоточный**
100	0	108
5100	87	31
10100	411	112
15100	1147	244
20100	2197	418
25100	3766	638
30100	6026	964
35100	9121	1280
40100	13160	1683
45100	18259	2179
50100	24616	2703
55100	32251	3239
60100	41435	3897
65100	52181	4575
70100	64554	5984
75100	79472	7617
80100	95870	8398
85100	114356	9400
90100	135587	11029
95100	162326	14553

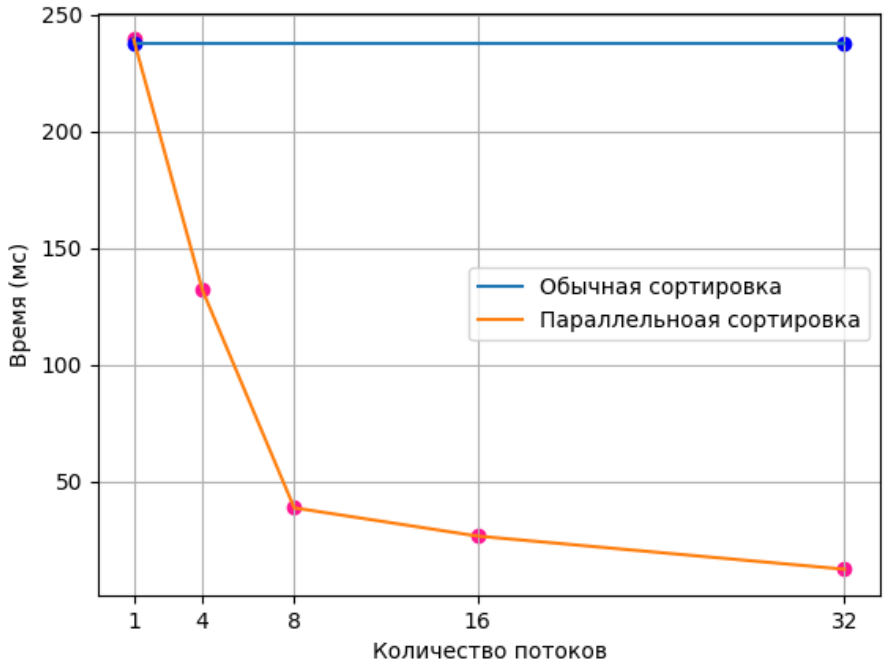

Рисунок 4.1 – Зависимость времени от кол-ва потока для массива размера 1000 элементов

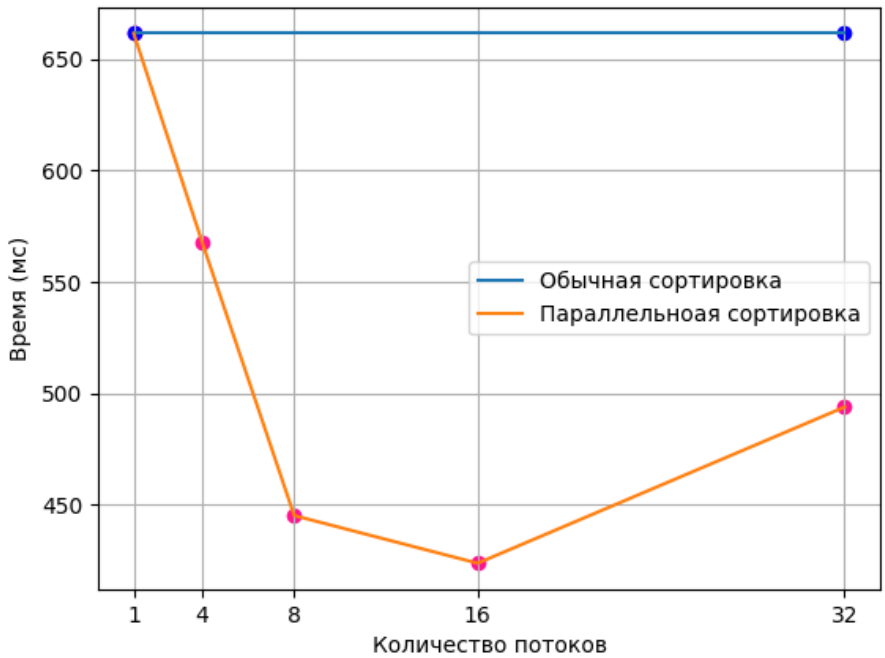

Рисунок 4.2 – Зависимость времени от кол-ва потока для массива размера 10000 элементов

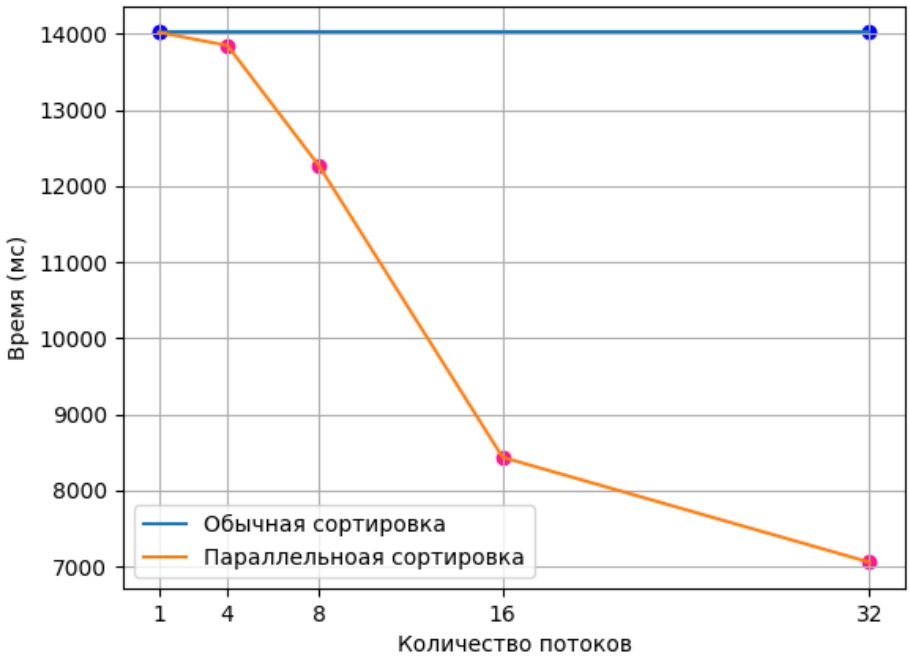

Рисунок 4.3 – Зависимость времени от кол-ва потока для массива размера 50000 элементов

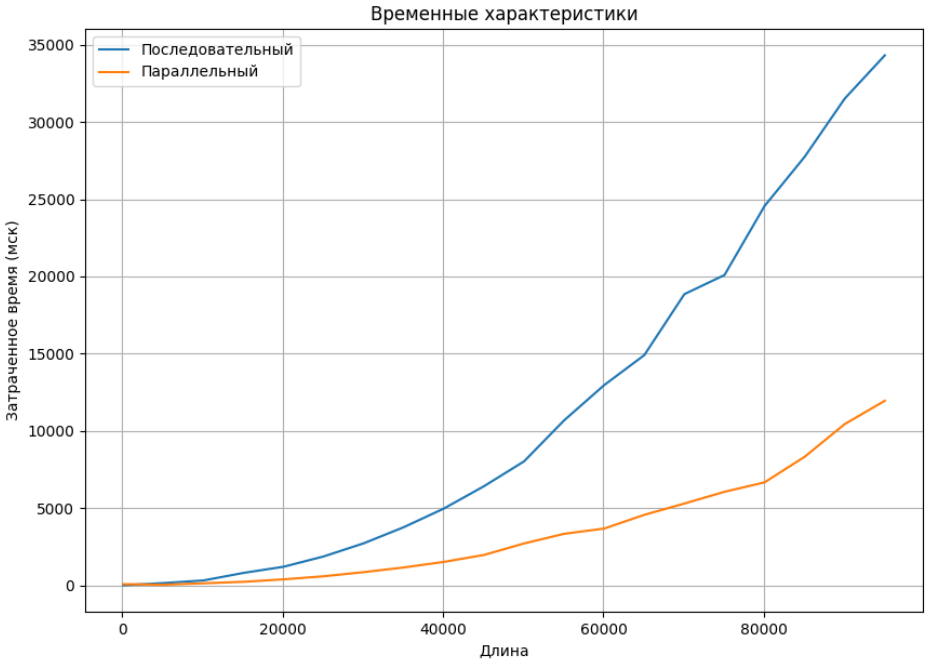

Рисунок 4.4 – Зависимость времени от длины массива для 4 потоков

Вывод

В данном разделе было произведено сравнение алгоритма ранговой сортировки при простой реализации и многопоточной. Результат показал, что выгоднее всего использовать все ядра процессора.

Заключение

В данной лабораторной работе были рассмотрены основополагающие материалы которые в дальнейшем потребовались при параллельной и однопоточной реализации алгоритма ранговой сортировки. Были рассмотрены схемы однопоточной и многопоточной реализации ранее разобранного алгоритма. Также были разобраны листинги, показывающие работу как однопоточного, так и многопоточного алгоритма ранговой сортировки.

В ходе выполнения лабораторной работы были решены следующие задачи:

1. Изучены основы параллельных вычислений
2. Применены изученные основы для реализации многопоточной сортировки.
3. Получены практические навыки.
4. Произведен сравнительный анализ параллельной и однопоточной реализации алгоритма сортировки.
5. Экспериментально подтверждены различия во временной эффективности реализации однопоточной и многопоточной ранговой сортировки.
6. Подготовлен отчет о лабораторной работе.

Поставленная цель достигнута.

Литература

[1] Mario Nemirovsky D. M. T. Multithreading Architecture // Morgan and Claypool Publishers. 2013.

[2] Olukotun K. Chip Multiprocessor Architecture — Techniques to Improve Throughput and Latency // Morgan and Claypool Publishers. 2007. p. 154.

[3] Фомин Э.С. Сортировка массивов коротких последовательностей. Институт Цитологии и Генетики СО РАН, 2010. с. 302.

[4] Документация по C# [Электронный ресурс]. Режим доступа: `https://docs.microsoft.com/ru-ru/dotnet/csharp/` (дата обращения: 30.09.2021).

[5] Visual Studio Microsoft [Электронный ресурс]. Режим доступа: `https://visualstudio.microsoft.com/ru/downloads/` (дата обращения: 30.09.2021).

[6] Windows [Электронный ресурс]. Режим доступа: `https://www.microsoft.com/ru-ru/windows` (дата обращения: 30.09.2021).

[7] Процессор Intel® Core™ i5-1135G7 [Электронный ресурс]. Режим доступа: `https://www.intel.ru/content/www/ru/ru/products/sku/208658/intel-core-i51135g7-processor-8m-cache-up-to-4-20-ghz/specifications.html` (дата обращения: 04.09.2021).