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 №5 по курсу "Анализ алгоритмов"

Тема Конвейерная обработка данных

Студент Криков А.В.

Группа ИУ7-53Б

Преподаватели Волкова Л.Л., Строганов Ю.В.

Москва — 2021 г.

Содержание

Введение

Конвейер — способ организации вычислений, используемый в современных процессорах и контроллерах с целью повышения их производительности (увеличения числа инструкций, выполняемых в единицу времени — эксплуатация параллелизма на уровне инструкций), технология, используемая при разработке компьютеров и других цифровых электронных устройств.

Сам термин «конвейер» пришёл из промышленности, где используется подобный принцип работы — материал автоматически подтягивается по ленте конвейера к рабочему, который осуществляет с ним необходимые действия, следующий за ним рабочий выполняет свои функции над получившейся заготовкой, следующий делает ещё что-то. Таким образом, к концу конвейера цепочка рабочих полностью выполняет все поставленные задачи, сохраняя высокий темп производства. Например, если на самую медленную операцию затрачивается одна минута, то каждая деталь будет сходить с конвейера через одну минуту. В процессорах роль рабочих исполняют функциональные модули, входящие в состав процессора.

Целью данной работы является изучение и программная реализация асинхронного взаимодействия потоков на примере конвейерной обработки данных.

В рамках выполнения работы необходимо решить следующие задачи:

- рассмотреть и изучить конвейерную обработку данных;
- привести схему реализации алгоритма конвейерной обработки данных;
- описать структуру разрабатываемого ПО;
- определить средства программной реализации;
- протестировать разработанное ПО;
- привести сведения о модулях программы;
- определить требования к ПО;
- исследовать и реализовать конвейер с количеством лент не меньше трех в многопоточной среде;

- на основании проделанной работы сделать выводы и подготовить отчет.

1 Аналитическая часть

В данном разделе будут рассмотрены алгоритмы: конвейерная обработка данных, кодирование строки.

1.1 Описание алгоритмов

1.1.1 Описание конвейерной обработки данных

Конвейеризация[1] - это техника, в результате которой задача или команда разбивается на некоторое число подзадач, которые выполняются последовательно. Каждая подкоманда выполняется на своем логическом устройстве. Все логические устройства (ступени) соединяются последовательно таким образом, что выход i-ой ступени связан с входом (i + 1)-ой ступни, все ступени работают одновременно. Множество ступеней называется конвейером. Выигрыш во времени достигается при выполнении нескольких задач за счет параллельной работы ступеней, вовлекая на каждом такте новую задачу или команду. В конвейере различают r последовательных этапов, так что когда i-я операция проходит s-й этап, то (i + k)-я операция проходит (s - k)-й этап.

1.1.2 Алгоритм кодирования строки

В данной лабораторной работе реализована некая функция кодирования строки, которая состоит из трех последовательных действий: применения функции шифра Цезаря[2], применения функции, меняющей регистр на противоположный, применение функции, меняющей местами элементы стоящие на n / 2 символов друг от друга, где n - размер строки. Если необходимо закодировать массив строк, то можно использовать конвейерную обработку данных. Таким образом задача будет решена эффективнее, чем при последовательном применении алгоритмов к массиву значений. Конвейер будет состоять из четырех уровней. Обработанные данные передаются последовательно с одного уровня (одной ленты) конвейера на

следующий (следующую ленту). Далее на каждом уровне осуществляется обработка данных. Для каждой ленты создается своя очередь задач, в которой хранятся все необработанные строки. На последнем уровне конвейера обработанные объекты попадают в пул обработанных задач.

Уровни конвейера:

1. Применение шифра Цезаря к строкам из первой очереди, запись результата во 2 очередь;

2. Применение функции, меняющей регистры символов к строкам 2 очереди, запись результата в 3 очередь;

3. Применение функции, меняющей местами символы в строке по определенному выше закону к строкам в 3 очереди, запись результата в пул обработанных задач.

Поскольку запись в очередь и извлечение из очереди это не атомарные операции, необходимо создать их таковыми, путем использования мьютексов (по одному на очередь), чтобы избежать ошибок в ситуации гонок.

Вывод

В данной работе стоит задача реализации алгоритма конвейерной обработки данных. Входными данными будет являтся число n - количество строк, которые необходимо закодировать. Выходными данными будет являтся массив закодированных строк. В связи с ограничениями накладываемыми на ПО, входное число n должно быть корректным. Необходимо дать теоретическую оценку последовательной и конвейерной реализации алгоритма кодирования строк.

2 Конструкторская часть

В данном разделе будут рассмотрены схемы рабочего и главного процессов, описаны способы тестирования, и определены структуры данных.

2.1 Разработка алгоритмов

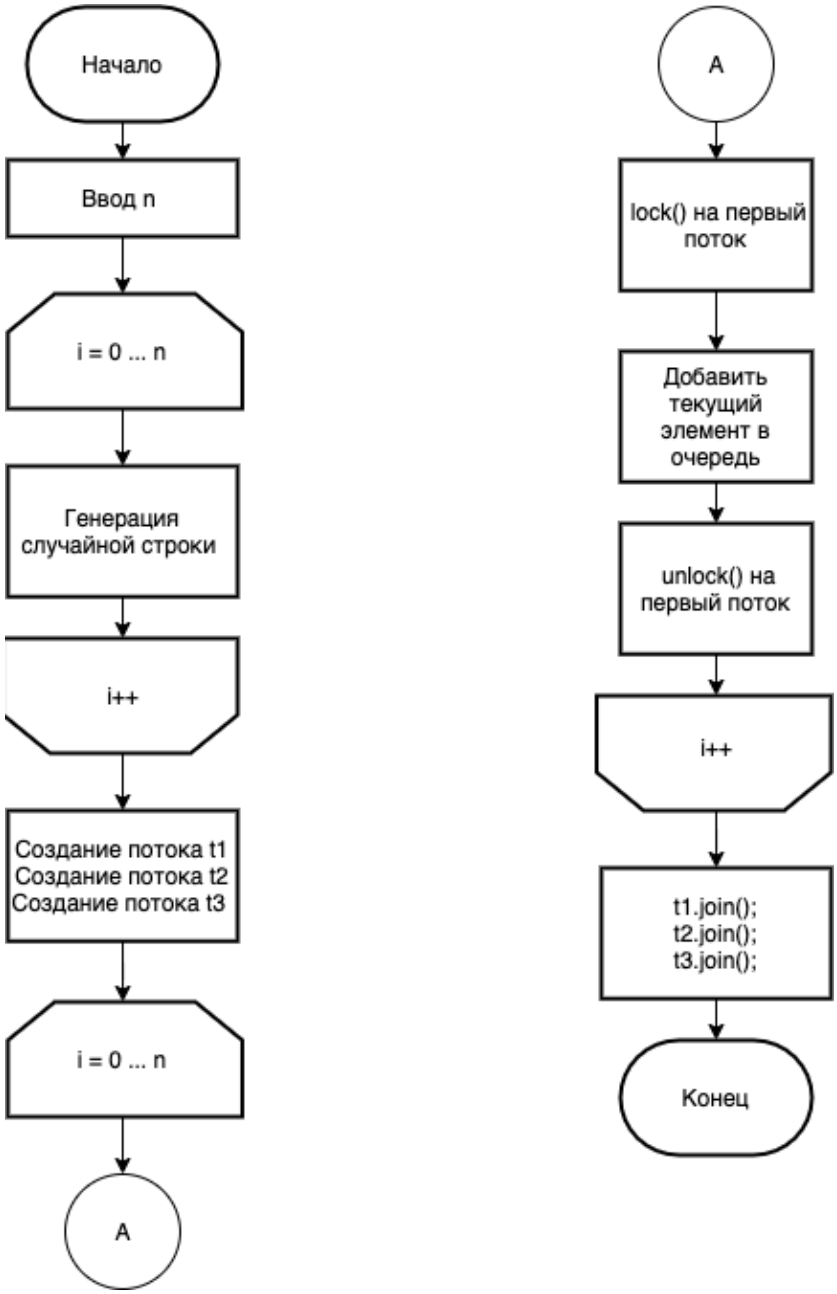

Рис. 2.1: Схема работы главного процесса

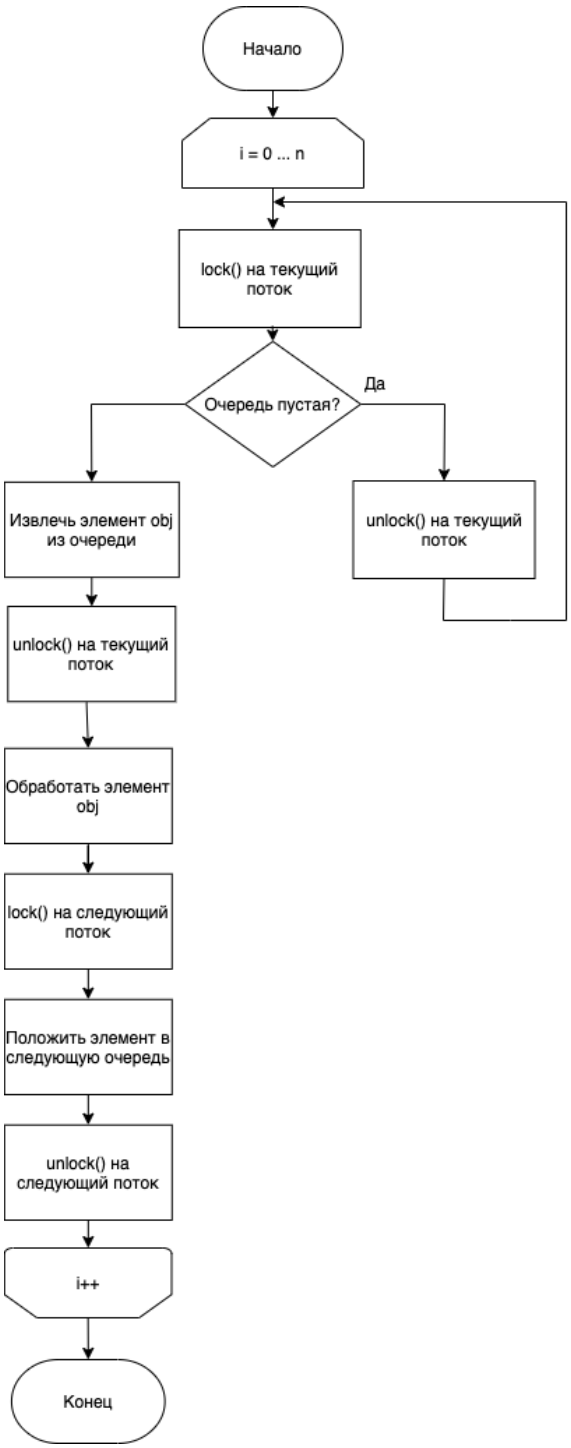

Рис. 2.2: Схема рабочего процесса

2.2 Описание структур данных

Исходя из условия задачи приходим к выводу о том, что для хранения объектов на каждой ступени конвейера необходимо использовать структуру данных - очередь. Для хранения слов, которые необходимо закодировать

наиболее удобно использовать структуру данных - массив строк.

2.3 Структура ПО

ПО будет состоять из следующих модулей:

- Основной модуль;
- Модуль включающий в себя реализацию алгоритма кодирования строки;
- Модуль для работы со ступенями конвейера;
- Модуль для работы с входными и выходными данными;
- Модуль логирования;

Вывод

На основе теоретических данных, полученных из аналитического раздела, были построены схемы главного и рабочего процессов. Приведена структура ПО, описаны структуры данных.

3 Технологическая часть

В данном разделе приведены средства реализации и листинг алгоритмов.

3.1 Средства реализации

Для разработки ПО был выбран язык Java, поскольку он предоставляет разработчику широкий спектр возможностей и позволяет разрабатывать кроссплатформенные приложени, а также предоставляет большой набор инструментов для работы с многопоточностью. В качестве среды разработки использовалась Visual Studio Code. [3]

3.2 Листинг кода

Листинг 3.1: Реализация первого уровня конвейера

```
1   public static void firstStage() {
2       while (!Main.queue1.isEmpty()) {
3           String curStr;
4           int curTask;
5           ConveyorObject conveyorObject;
6
7           synchronized (Main.queue1) {
8               conveyorObject = Main.queue1.poll();
9           }
10
11          curStr = conveyorObject.string;
12          curTask = conveyorObject.taskNum;
13
14          long convTimeStart = System.nanoTime();
15          String newStr = Encryption.reverse(curStr);
16          Main.timer.addTimeFirstConv(curStr, System.nanoTime() - convTimeStart);
17          Main.timer.logCurEvent(curTask, true, System.nanoTime());
18
19          synchronized (Main.queue2) {
20              Main.queue2.add(new ConveyorObject(newStr, curTask, System.nanoTime()));
21          }
22      }
23  }
```

Листинг 3.2: Реализация второго уровня конвейера

```
1   public static void secondStage() {
2       do {
3           if (!Main.queue2.isEmpty()) {
4               String temp;
5               int curTask;
6               ConveyorObject obj;
7
8               synchronized (Main.queue2) {
9                   obj = Main.queue2.poll();
10              }
11
12              temp = obj.string;
13              curTask = obj.taskNum;
14
15              long convTimeStart = System.nanoTime();
16              String newStr = Encryption.fromLowerToUpper(temp);
17              Main.timer.addTimeSecondConv(curTask, System.nanoTime() - convTimeStart);
18
19              synchronized (Main.queue3) {
20                  Main.queue3.add(new ConveyorObject(newStr, curTask,
                        System.nanoTime()));
21              }
22          }
23      } while (!Main.queue1.isEmpty() || !Main.queue2.isEmpty());
24  }
```

Листинг 3.3: Реализация третьего уровня конвейера

```
1   public static void thirdStage() {
2       do {
3           if (!Main.queue3.isEmpty()) {
4               String temp;
5               int curTask;
6               ConveyorObject obj;
7
8               synchronized (Main.queue3) {
9                   obj = Main.queue3.poll();
10              }
11
12              temp = obj.string;
13              curTask = obj.taskNum;
14
15              long convTimeStart = System.nanoTime();
16              String newStr = Encryption.caesar(temp);
17              Main.timer.addTimeThirdConv(curTask, System.nanoTime() - convTimeStart);
18              Main.timer.logCurEvent(curTask, false, System.nanoTime());
19
```

```
            synchronized (Main.res) {
                Main.res.add(newStr);
            }
        }
    } while (!Main.queue1.isEmpty() || !Main.queue2.isEmpty() ||
        !Main.queue3.isEmpty());
}
```

Листинг 3.4: Реализация главного рабочего процесса

```
public static void main(String[] args) throws InterruptedException,
    FileNotFoundException {
    System.out.print("Input␣amount␣of␣words:␣");
    Scanner scanner = new Scanner(System.in);
    n = scanner.nextInt();

    queue1 = new ArrayBlockingQueue<>(n);
    queue2 = new ArrayBlockingQueue<>(n);
    queue3 = new ArrayBlockingQueue<>(n);

    for (int i = 0; i < n; i++) {
        String s = Utils.generate();
        words.add(s);
    }

    timer = new Timer();
    timer.setSize(n);

    Thread threadInit = new Thread(Main::init);
    threadInit.start();
    threadInit.join();

    startTime = System.nanoTime();

    Thread thread1 = new Thread(ConveyorStages::firstStage);
    Thread thread2 = new Thread(ConveyorStages::secondStage);
    Thread thread3 = new Thread(ConveyorStages::thirdStage);

    thread3.start();
    thread2.start();
    thread1.start();

    thread1.join();
    thread2.join();
    thread3.join();

    endTime = System.nanoTime();
    timer.addConvTime(endTime - startTime);

```

```
        runLinear();

        timer.printLog();
    }
```

Листинг 3.5: Реализация алгоритма кодирования строки

```
    public static String caesar(String s) {
        StringBuilder s_array = new StringBuilder(s);
        for (int i = 0; i < s.length(); i++) {
            if (s_array.charAt(i) == 'z') {
                s_array.setCharAt(i, 'a');
            } else if (s_array.charAt(i) == 'Z') {
                s_array.setCharAt(i, 'A');
            } else {
                s_array.setCharAt(i, (char) (s_array.charAt(i) + 1));
            }
        }
        return s_array.toString();
    }

    public static String fromLowerToUpper(String s) {
        StringBuilder s_array = new StringBuilder(s);

        for (int i = 0; i < s.length(); i++) {
            if (Character.isLowerCase(s_array.charAt(i))) {
                s_array.setCharAt(i, Character.toUpperCase(s_array.charAt(i)));
            } else {
                s_array.setCharAt(i, Character.toLowerCase(s_array.charAt(i)));
            }
        }

        return s_array.toString();
    }

    public static String reverse(String s) {
        int shift = s.length() / 2;
        StringBuilder s_array = new StringBuilder(s);

        for (int i = 0; i < s.length() / 2; i++) {
            char st = s_array.charAt(i);
            s_array.setCharAt(i, s_array.charAt(i + shift));
            s_array.setCharAt(i + shift, st);
        }

        return s_array.toString();
    }
```

3.3 Тестирование функций

В таблице 3.1 приведены тесты для функций, реализующих алгоритм кодирования строки. Тесты пройдены успешно.

Количество строк	Исходные строки	Ожидаемый результат
2	$\begin{pmatrix} abd & bcv \end{pmatrix}$	$\begin{pmatrix} cbe & cdw \end{pmatrix}$
-100	abcdeijfladkk	Некорректное количество строк

Таблица 3.1: Тестирование функций

Вывод

Правильный выбор инструментов разработки позволил эффективно реализовать алгоритмы, настроить модульное тестирование и выполнить исследовательский раздел лабораторной работы.

4 Исследовательская часть

В данном разделе будет произведено сравнение последовательной реализации алгоритма кодирования строк и конвейера с использованием многопоточности, а также будет приведена демонстрация работы программы.

4.1 Демонстрация работы программы

На вход программе подается количество строк. Вывод программы представлен на рисунке 4.1 .

```
Input amount of words: 3



                              Сравнение линейной и конвейерной реализации
                                  Линейная                  Конвейерная
Общее время работы системы:         161988340                  62303208
Среднее время обработки заявки:     53996113                   19521671

 Лог программы
Task = 1 Start   Part = 1   Time = 135361333
Task = 1 End     Part = 1   Time = 140720208
Task = 1 Start   Part = 2   Time = 153290250
Task = 2 Start   Part = 1   Time = 157041166
Task = 1 End     Part = 2   Time = 159736625
Task = 1 Start   Part = 3   Time = 159784291
Task = 2 End     Part = 1   Time = 162138166
Task = 1 End     Part = 3   Time = 164528916
Task = 2 Start   Part = 2   Time = 174660916
Task = 3 Start   Part = 1   Time = 178418333
Task = 3 End     Part = 1   Time = 179169875
Task = 2 End     Part = 2   Time = 179610500
Task = 2 Start   Part = 3   Time = 179672250
Task = 2 End     Part = 3   Time = 187202583
Task = 3 Start   Part = 2   Time = 192183708
Task = 3 End     Part = 2   Time = 192468791
Task = 3 Start   Part = 3   Time = 192504291
Task = 3 End     Part = 3   Time = 192656750
```

Рис. 4.1: Результат работы программы

4.2 Технические характеристики

- Операционная система: Windows 10. [4]
- Память: 16 GiB.
- Процессор: Intel(R) Core(TM) i7-4700HQ CPU @ 2.40GHz. [5]

4.3 Временные характеристики

Для сравнения возьмем количество строк равное: [5, 10, 15, ... 100]. Так как кодирование исходной строки считается короткой задачей, возспользуемся усреднением массового эксперимента. Для этого сложим результат работы алгоритма N раз (N >= 10), после чего поделим на N. Тем самым получим достаточно точные характеристики времени. Результат сравнения последовательной реализации трех алгоритмов кодирования строки и конвейера с использованием многопоточности представлен на рис 4.1.

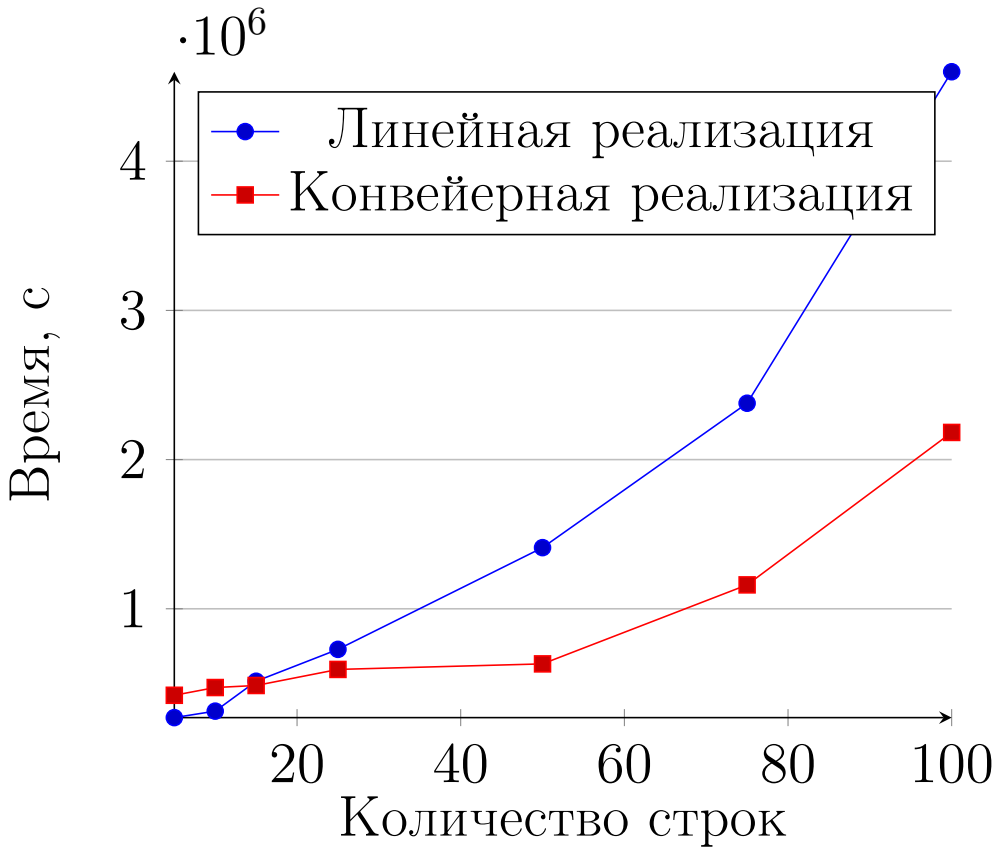

Рис. 4.2: Зависимость времени работы алгоритмов от количества исходных строк

Вывод

В результате эксперимента было получено, что время исполнения конвейерной реализации значительно меньше, чем исполнение последовательной реализации. В начале последовательная реализация выигрывает по времени, поскольку тратится время на создание потоков и ожидание доступа к переменной, что значительно снижает работоспособность. Так при количестве строк = 100 конвейерная реализация алгоритма кодирования строк быстрее последовательного выполнения алгоритма приблизительно в 2.5 раза. Можно сделать вывод, что для получения большей производительности стоит использовать конвейерную реализацию алгоритма.

Заключение

В рамках выполнения данной лабораторной работы были достигнуты следующие цели:

- исследованы основы конвейерных вычислений;
- исследованы основные методы организации конвейерных вычислений;
- проведено сравнение существующих методов организации конвейерных вычислений;
- определены средства программной реализации;
- определены требования к ПО;
- приведены сведения о модулях ПО;
- приведены экспериментальные замеры временных характеристик реализованного конвейера;
- подготовлен отчет по проделанной работе.

Конвейерная обработка данных - полезный инструмент, который уменьшает время выполнения программы за счет параллелльной обработки данных. Метод дает выйгрыш по времени в том случае, когда выполняемые задачи намного больше по времени, чем время, затрачиваемое на реализацию конвейера.

Список литературы

[1] Параллельная обработка данных. Режим доступа: https://parallel.ru/vvv/lec1.html.

[2] Шифр цезаря. Режим доступа: https://kmb.cybber.ru/crypto/caezar_cipher/main.html.

[3] Documentation for Visual Studio Code [Электронный ресурс]. Режим доступа: https://code.visualstudio.com/docs.

[4] Клиентская документация по Windows для ИТ-специалистов [Электронный ресурс]. Режим доступа: https://docs.microsoft.com/ru-ru/windows/resources/.

[5] Процессор Intel® Core™ i7-4700HQ [Электронный ресурс]. Режим доступа: https://ark.intel.com/content/www/ru/ru/ark/products/122589/intel-core-i7-8550u-processor-8m-cache-up-to-4-00-ghz.html.