МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ   
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное  
образовательное учреждение высшего образования  
«Самарский национальный исследовательский университет  
имени академика С.П. Королева»  
(Самарский университет)

Институт информатики, математики и электроники

Факультет информатики

Кафедра программных систем

**ОТЧЁТ**

по лабораторной работе №3 по дисциплине

«Моделирование и анализ параллельных алгоритмов»

Студент группы №6132-020402D В.А. Артамонов

Проверил Д.С. Оплачко

Самара 2021

СОДЕРЖАНИЕ

[1 Постановка задачи 3](#_Toc87634674)

[2 Описание работы параллельного алгоритма 5](#_Toc87634675)

[3 Текст различных вариантов программы 6](#_Toc87634676)

[3.1 Параллельный алгоритм 6](#_Toc87634677)

[3.2 Последовательный алгоритм 8](#_Toc87634678)

[4 Результаты вычислительных экспериментов 9](#_Toc87634679)

[5 Заключение 12](#_Toc87634680)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 13](#_Toc87634681)

[ПРИЛОЖЕНИЕ А 14](#_Toc87634682)

[Код проверочного класса 14](#_Toc87634683)

[ПРИЛОЖЕНИЕ Б 16](#_Toc87634684)

[Код класса мейн 16](#_Toc87634685)

[ПРИЛОЖЕНИЕ В 17](#_Toc87634686)

[Код параллельного алгоритма 17](#_Toc87634687)

[ПРИЛОЖЕНИЕ Г 18](#_Toc87634688)

[Код последовательного алгоритма 18](#_Toc87634689)

1. Постановка задачи
2. С использованием одной из моделей взаимодействия параллельных процессов разработать параллельную программу нахождения всех простых чисел в диапазоне от 1 до *N*. Целое число *x*>1 называется простым, если оно делится без остатка только на x и на 1. Любое целое число *y*>1 можно представить в виде произведения простых чисел.
3. Программа должна работать с любым количеством потоков, задаваемым пользователем. Программа должна реализовать следующий алгоритм:

* Диапазон [1, *N*] разбивается на *P* частей, где *P* – количество потоков или процессов параллельной программы.
* Каждый поток находит простые числа в своем диапазоне, последовательно проверяя все числа внутри диапазона.
* Все потоки используют общий буфер (одномерный массив) для хранения найденных простых чисел. Каждый поток при нахождении очередного простого числа добавляет его в буфер.
* Определение того, является очередное проверяемое число простым или нет, выполняется его делением на уже найденные простые числа, находящиеся в общем буфере. При этом возникает задача синхронизации: при анализе очередного числа *y* поток должен убедиться, что буфер содержит все числа, меньшие *y*, которые могут быть его делителями. Поскольку некоторые из этих чисел добавляются в буфер другими потоками, текущий поток должен приостановиться и подождать, когда остальные потоки добавят все необходимые ему числа.

3) Измерить длительность работы программы для различного количества потоков. Число *N* выбрать таким, чтобы последовательная программа работала не менее минуты. Количество потоков взять от 1 до 64, используя до 10 различных значений.

4) Добиться того, чтобы параллельная программа имела ускорение больше 1.

5) Объяснить наблюдаемые изменения длительности работы программы при изменении количества потоков от 1 до *P*.

6) Составить отчет по результатам работы.

1. Описание работы параллельного алгоритма

Принцип многопоточности, написанный в лабораторной работе № 3 на языке Java, представлен в виде класса Parallel [1]. Parallel, в свою очередь, делится на 2 конструктора:

primes – сначала выделяет первичные простые числа, после создает массив и выделяет для каждого потока его диапазон, и потом обращаясь в класс ProverkaNaProstoe задает общий принцип работы каждому потоку, а также выводит затраченное время.

ranges – распределяет все числа из диапазона между заданным количеством потоков [2].

Принцип работы следующий: задаются первоначальные данные (общий диапазон чисел и количество потоков), далее запускается параллельный алгоритм, в котором задаются первичные простые числа, после происходит разделение на потоки и их диапазоны, которые записываются в общий буфер. Далее каждый поток ожидает простые числа предыдущего потока и начинает делить каждое свое число из диапазона на все числа, записанные в буфер, по пути заходя в класс ProverkaNaProstoe, чтобы проверить принадлежность рассматриваемого числа к числу «простых». Таким образом заполняется весь буфер и все потоки работают относительно предыдущих потоков (могут уйти в ожидание, при помощи метода wait(), пока не получат все простые числа «делители»).

1. Текст различных вариантов программы
   1. Параллельный алгоритм

public class Parallel {  
  
 private static final int *MinProstoeChislo* = 2;  
 private static final int *MinPotok* = 1;  
 private static final int *MaxPotok* = 64;  
  
 public static ArrayList<Integer> primes(final int lastNumber, final int threadsNum) {  
  
 if (lastNumber < *MinProstoeChislo*) {  
 throw new IndexOutOfBoundsException();  
 }  
 if (threadsNum < *MinPotok* || threadsNum > *MaxPotok*) {  
 throw new IllegalArgumentException();  
 }  
  
 List<Thread> threads = new ArrayList<>(threadsNum);  
  
 ArrayList<Integer> primes = new ArrayList<Integer>(lastNumber / 2);  
 primes.add(2);  
  
 int[][] ranges = *ranges*(lastNumber, threadsNum);  
  
 long startMillis = System.*currentTimeMillis*();  
 Thread currThread;  
 for (int[] range : ranges) {  
 currThread = new ProverkaNaProstoe(primes, range[0], range[1]);  
 currThread.start();  
 threads.add(currThread);  
 }  
  
 for (Thread thread : threads) {  
 try {  
 thread.join();  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
 long finishMillis = System.*currentTimeMillis*();  
 long consumedMillis = finishMillis - startMillis;  
 System.*out*.println("Consumed millis: " + consumedMillis);  
 return primes;  
 }  
  
 private static int[][] ranges(int lastNumber, int threadsNum) {  
  
 float lowLimit = 2.0f;  
 float partition = (lastNumber - lowLimit) / threadsNum;  
 if (partition < 1.0) {  
 throw new IllegalArgumentException();  
 }  
  
 int[][] ranges = new int[threadsNum][2];  
  
 ranges[0][0] = Math.*round*(lowLimit);  
 for (int rowIndex = 0; rowIndex < threadsNum - 1; rowIndex++) {  
 lowLimit += partition;  
 ranges[rowIndex][1] = Math.*round*(lowLimit);  
 ranges[rowIndex + 1][0] = ranges[rowIndex][1] + 1;  
 }  
 ranges[threadsNum - 1][1] = lastNumber;  
  
 return ranges;  
 }  
}

* 1. Последовательный алгоритм

import java.util.ArrayList;  
  
public class Posled {  
  
 public static ArrayList<Integer> primes(int lastNumber) {  
  
 if (lastNumber < 2) {  
 throw new IndexOutOfBoundsException();  
 }  
  
 ArrayList<Integer> primes = new ArrayList<>(lastNumber / 2);  
  
 long startMillis = System.*currentTimeMillis*();  
 primes.add(2);  
 for (int i = 3; i <= lastNumber; i += 2) {  
 if (*isPrime*(i, primes)) {  
 primes.add(i);  
 }  
 }  
 long finishMillis = System.*currentTimeMillis*();  
 long consumedMillis = finishMillis - startMillis;  
 System.*out*.println("Consumed millis: " + consumedMillis);  
 return primes;  
 }  
 private static boolean isPrime(int number, ArrayList<Integer> primesBeforeNumber) {  
 double sqrtedNumber = Math.*sqrt*(number);  
 for (int prime : primesBeforeNumber) {  
 if (prime > sqrtedNumber) {  
 return true;  
 }  
 if (number % prime == 0) {  
 return false;  
 }  
 }  
 return true;  
 }  
}

1. Результаты вычислительных экспериментов

Вычислительные эксперименты, представленные в данном разделе, проводились на вычислительной системе с процессором, обладающим характеристиками, представленными в таблице 1.

Таблица 1 – Характеристики вычислительной системы

|  |  |
| --- | --- |
| Модель процессора | Intel(R) Core(TM) i3-6006U |
| CPU | 2.00GHz |
| Число ядер | 4 |
| Hyper-Threading (multithreading) | Есть |

В ходе выполнения лабораторной работы были проведены измерения времени работы последовательного и параллельного алгоритмов. Результаты представлены в таблице 2.

Таблица 2 – Результаты работы алгоритмов

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Кол-во потоков | Кол-во  чисел | Время работы последова-тельного алгоритма 𝑇П, мс | Время работы паралле-льного алгоритма 𝑇||, мс |  |
| 10 | 2\*10^8 | 102634 | 32614 | 3,147 |
| 16 | 106789 | 32437 | 3,292 |
| 22 | 107452 | 28569 | 3,758 |
| 28 | 100687 | 30534 | 3,298 |
| 34 | 106498 | 31284 | 3,404 |
| 40 | 107729 | 27262 | 3,952 |
| 46 | 110435 | 32404 | 3,408 |
| 52 | 106508 | 31515 | 3,38 |
| 58 | 106395 | 29385 | 3,621 |
| 64 | 106473 | 28999 | 3,672 |

Полученные данные можно представить в виде графиков.

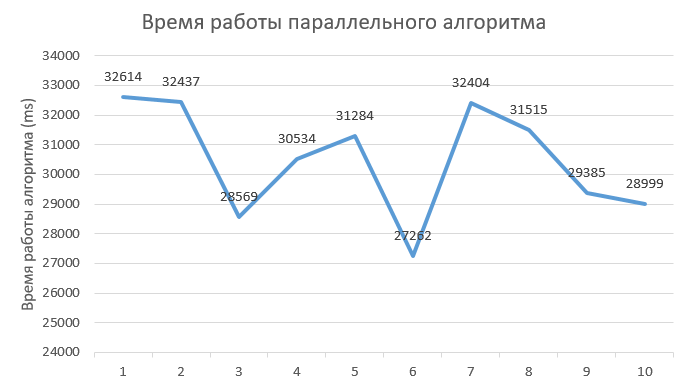
На рисунке 1 представлен график зависимости времени работы последовательного алгоритма от размера файла и количества строк.

Рисунок 1 – Зависимость времени работы последовательного алгоритма от размера файла и количества строк



На рисунке 2 представлен график зависимости времени работы параллельного алгоритма от размера файла и количества строк.

Рисунок 2 – Зависимость времени работы параллельного алгоритма от размера файла и количества строк



На рисунке 3 представлен график ускорения работы программы.



Рисунок 3 – Ускорение работы программы

1. Заключение

По полученным данным видно, что затраченное время на работу алгоритма почти не различается, как в параллельном, так и в последовательном. Из полученных данных видно, что параллельный алгоритм во всех случаях работает быстрее. Наибольшего ускорения программа достигает, когда количество потоков равно 40 [3].

Параллельная программа работает быстрее чем последовательная, за счет того, что общий диапазон [1, *N*] распределен между потоками. Не смотря на то, что присутствует лишняя задержка в параллельном алгоритме, за счет методов synchronized и wait [4].

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Многопоточность в Java/Хабр. [Электронный ресурс] URL: https://habr.com/ru/post/164487/ (дата обращения: 26.10.2021).
2. Поиск и проверка простых чисел на Java. [Электронный ресурс] URL: https://javascopes.com/find-check-prime-numbers-in-java-81de4b4e/ (дата обращения: 12.11.2021).
3. Воеводин В.В., Воеводин Вл.В. Параллельные вычисления СПб.:БХВ-Петербург, 2002. - 608 с.
4. Грегори Р. Эндрюс Основы многопоточного, параллельного и распределенного программирования - Вильямс, 2002. - 512 с.

ПРИЛОЖЕНИЕ А

Код проверочного класса

import java.util.ArrayList;  
  
public class ProverkaNaProstoe extends Thread {  
  
 private final ArrayList<Integer> primes;  
 private final int LevayaGranica;  
 private final int PravayaGranica;  
  
 public ProverkaNaProstoe(ArrayList<Integer> primes, int LevayaGranica, int PravayaGranica) {  
  
 if (LevayaGranica > PravayaGranica) {  
 throw new IllegalArgumentException();  
 }  
  
 this.primes = primes;  
 this.LevayaGranica = LevayaGranica;  
 this.PravayaGranica = PravayaGranica;  
 }  
  
 @Override  
 public void run() {  
  
 int startNumber = (LevayaGranica % 2 == 0) ? LevayaGranica + 1 : LevayaGranica;  
  
 for (int i = startNumber; i <= PravayaGranica; i += 2) {  
 if (isPrime(i)) {  
 synchronized (primes) {  
 primes.add(i);  
 primes.notifyAll();  
 }  
 }  
 }  
  
 System.*out*.println(Thread.*currentThread*().getName() + " finished " + "[" + LevayaGranica + "; " + PravayaGranica + "]");  
 }  
  
 private boolean isPrime(int number) {  
  
 double sqrtedNumber = Math.*sqrt*(number);// в каждом диапазоне каждое нечетное число будет проверяться на простое  
 //делением на предыдущие простые числа отобранные через корень из самого числа  
 synchronized (primes) {  
 while (primes.get(primes.size() - 1) < (int) sqrtedNumber) {  
 try {  
 primes.wait();  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
 }  
  
 for (int i = 0; i < primes.size(); i++) {  
 if (primes.get(i) < number) {  
 if (primes.get(i) > sqrtedNumber) {  
 return true;  
 }  
  
 if (number % primes.get(i) == 0) {  
 return false;  
 }  
 }  
 }  
  
 return true;  
 }  
}

ПРИЛОЖЕНИЕ Б

Код класса мейн

import java.util.ArrayList;  
  
public class Main {  
  
 private static final int *KolichestvoChisel* = 10000;  
 private static final int *KolichestvoPotokov* = 10;  
  
 public static void main(String[] args) {  
 System.*out*.println("Лабораторная работа №.3---Выполнил студент группы 6132 Артамонов В.А.");  
 Posled.*primes*(*KolichestvoChisel*);  
 ArrayList<Integer> primes1 = Parallel.*primes*(*KolichestvoChisel*, *KolichestvoPotokov*);  
  
 System.*out*.println(primes1);  
 }  
}

ПРИЛОЖЕНИЕ В

Код параллельного алгоритма

import java.util.ArrayList;  
public class Posled {  
 public static ArrayList<Integer> primes(int lastNumber) {  
 if (lastNumber < 2) {  
 throw new IndexOutOfBoundsException();  
 }  
 ArrayList<Integer> primes = new ArrayList<>(lastNumber / 2);  
 long startMillis = System.*currentTimeMillis*();  
 primes.add(2);  
 for (int i = 3; i <= lastNumber; i += 2) {  
 if (*isPrime*(i, primes)) {  
 primes.add(i);  
 }  
 }  
 long finishMillis = System.*currentTimeMillis*();  
 long consumedMillis = finishMillis - startMillis;  
 System.*out*.println("Consumed millis: " + consumedMillis);  
  
 return primes;  
 }  
  
 private static boolean isPrime(int number, ArrayList<Integer> primesBeforeNumber) {  
  
 double sqrtedNumber = Math.*sqrt*(number);  
  
 for (int prime : primesBeforeNumber) {  
  
 if (prime > sqrtedNumber) {  
 return true;  
 }  
  
 if (number % prime == 0) {  
 return false;  
 }  
 }  
  
 return true;  
 }  
}

ПРИЛОЖЕНИЕ Г

Код последовательного алгоритма

import java.util.ArrayList;  
  
public class Posled {  
  
 public static ArrayList<Integer> primes(int lastNumber) {  
  
 if (lastNumber < 2) {  
 throw new IndexOutOfBoundsException();  
 }  
  
 ArrayList<Integer> primes = new ArrayList<>(lastNumber / 2);  
  
 long startMillis = System.*currentTimeMillis*();  
 primes.add(2);  
 for (int i = 3; i <= lastNumber; i += 2) {  
 if (*isPrime*(i, primes)) {  
 primes.add(i);  
 }  
 }  
 long finishMillis = System.*currentTimeMillis*();  
 long consumedMillis = finishMillis - startMillis;  
 System.*out*.println("Consumed millis: " + consumedMillis);  
  
 return primes;  
 }  
  
 private static boolean isPrime(int number, ArrayList<Integer> primesBeforeNumber) {  
  
 double sqrtedNumber = Math.*sqrt*(number);  
  
 for (int prime : primesBeforeNumber) {  
  
 if (prime > sqrtedNumber) {  
 return true;  
 }  
  
 if (number % prime == 0) {  
 return false;  
 }  
 }  
  
 return true;  
 }  
}