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 №1 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](#_Toc85548064)

[2 Описание работы параллельного алгоритма 4](#_Toc85548065)

[3 Текст различных вариантов программы 5](#_Toc85548066)

[3.1 Параллельный алгоритм 5](#_Toc85548067)

[3.2 Последовательный алгоритм 6](#_Toc85548068)

[4 Результаты вычислительных экспериментов 6](#_Toc85548069)

[5 Заключение 9](#_Toc85548070)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 11](#_Toc85548071)

[ПРИЛОЖЕНИЕ А 12](#_Toc85548072)

[Код параллельной программы 12](#_Toc85548073)

[ПРИЛОЖЕНИЕ Б 16](#_Toc85548074)

[Код последовательной программы 16](#_Toc85548075)

1. Постановка задачи

1) Разработать параллельную программу, выполняющую копирование данных из одного массива (A) в другой массив (B) с использованием промежуточного буфера. Программа должна состоять из двух потоков. Первый поток – производитель – содержит исходный массив А и копирует данные из него в буфер по одному элементу. Второй поток содержит массив В и производит чтение данных из буфера по одному элементу и запись их в массив В. Запись в буфер возможна при наличии в нем хотя бы одного свободного элемента. Чтение из буфера возможно при наличии в нем хотя бы одного несчитанного элемента. С буфером связаны два счетчика: первый счетчик содержит индекс первого доступного элемента для чтения (начало данных), второй счетчик содержит индекс первого доступного для записи элемента. Если счетчики равны, буфер пуст. Буфер циклический, то есть второй счетчик может быть меньше первого. Если второй счетчик меньше первого на 1, то буфер заполнен. Альтернативным подходом может служить учет только занятых элементов буфера. В этом случае первый счетчик содержит индекс первого занятого элемента, второй счетчик содержит индекс последнего занятого элемента.

2) Измерить время работы программы для различных значений длины буфера. Размер массивов должен быть таким, чтобы массивы A, B и буфер заполняли максимум свободной оперативной памяти. Минимальный размер массива A – 108 элементов. Размер буфера – от 10 элементов до половины длины массива A. Рекомендуется использовать не менее 10 различных значений размера буфера при проведении вычислительных экспериментов.

3) Объяснить наблюдаемые изменения длительности работы программы.

4) Сравнить время работы параллельной программы с временем работы ее последовательного аналога, осуществляющего последовательное копирование путем чередования фаз заполнения и освобождения буфера.

5) Объяснить наблюдаемые закономерности.

6) Составить отчет по результатам работы.

1. Описание работы параллельного алгоритма

Принцип многопоточности, написанной лабораторной работы № 1 на языке Java, представлен в виде класса Thread(поток). Thread, в свою очередь, делится на два потока: Producer (Производитель) и Consumer (Потребитель)[1].

Producer осуществляет запись данных из массива А в буфер, а Consumer считывает данные из буфера и записывать их в массив В. Взаимодействие массивов осуществляется при помощи буфера, в котором содержатся, согласно заданию, два счетчика: первого и последнего занятых элементов, так как мы работает с несколькими потоками, то они имеют модификатор volatile.

Принцип работы, следующий: потоки запускаются одновременно при помощи метода start (), поток чтения (выполненный при помощи метода read ()) ожидает, пока буфер не будет полностью заполнен при помощи потока записи. Как только заполняется буфер – поток чтения начинает свою работу до тех пор, пока в буфере не останется элементов. Поток записи (выполненный при помощи метода write ()) в этот момент ожидает. После чего снова начинает работать поток записи и т.д. Для обеспечения синхронизации были использованы методы реализации блокировок – wait () и notify (). Для того чтобы дождаться завершения работы потока используется метод join (), блокирующий главный вызывающий поток до завершения запущенного им потока [2].

1. Текст различных вариантов программы
   1. Параллельный алгоритм

public class Main {  
 public static void main(String[] args) throws InterruptedException {  
 System.out.println("Лабораторная работа номер 1. Выполнил: Артамонов Владислав 6132");  
 int massive\_length = 100\_000\_000;  
 Scanner scan = new Scanner(System.in);  
 System.out.println("Длина массива 10^8 элементов.");  
 System.out.print("Введите длину буфера: ");  
 int buffer\_length = scan.nextInt();  
 int[] arr1 = new int[massive\_length];  
 int[] arr2 = new int[massive\_length];  
 Buffer buffer = new Buffer(buffer\_length);  
 long totalTime = 0;  
 int cycles = 5;  
 Random random = new Random(System.currentTimeMillis());  
 for (int j = 0; j < cycles; j++) {  
  
  
 for (int i = 0; i < massive\_length; i++) {  
 arr1[i] = random.nextInt();  
 }  
  
 Producer producer = new Producer(arr1, buffer);  
 Consumer consumer = new Consumer(arr2, buffer);  
 Thread producer\_thread = new Thread(producer);  
 Thread consumer\_thread = new Thread(consumer);  
  
 long start = System.currentTimeMillis();  
 producer\_thread.start();  
 consumer\_thread.start();  
 consumer\_thread.join();  
 long finish = System.currentTimeMillis();  
  
  
 long timeConsumedMillis = (finish - start) /\*/ cycles\*/;  
 totalTime += timeConsumedMillis;  
 System.out.println(j + ") Время работы параллельной программы: " + timeConsumedMillis + " мс");  
 System.out.println(Arrays.equals(arr1, arr2));  
 }  
 System.out.println("Среднее время работы параллельной программы: " + totalTime / cycles + " мс");  
 }  
}

* 1. Последовательный алгоритм

public class posled {  
 public static void main(String[] args) {  
 System.*out*.println("Лабораторная работа номер 1. Выполнил: Артамонов Владислав 6132");  
 int massive\_length = 100\_000\_000;  
 Scanner scan = new Scanner(System.*in*);   
 System.*out*.println("Длина массива - 10^8 элементов.");  
 System.*out*.print("Введите размер буфера: ");  
 int buffer\_length = scan.nextInt();  
 int[] arr1 = new int[massive\_length];  
 for (int i = 0; i < massive\_length; i++) {  
 arr1[i] = i;  
 }  
 int[] arr2 = new int[massive\_length];  
 int[] buffer = new int[buffer\_length];  
 int pointer = 0;  
 int counter\_cycles = 5;  
 long start = System.*currentTimeMillis*();  
 for (int i = 0; i < counter\_cycles; i++) {  
 while (pointer + buffer\_length <= massive\_length) {  
 for (int j = 0; j < buffer\_length; j++) {

buffer[j] = arr1[j + pointer];

}

for ( int j = 0; j < buffer\_length; j++) {

arr2[j + pointer] = buffer[j];

}  
 pointer = pointer + buffer\_length;  
 }  
 pointer = 0;  
 }  
 long finish = System.*currentTimeMillis*();  
 long timeConsumedMillis = (finish - start) / counter\_cycles;  
 System.*out*.println("Время работы : " + timeConsumedMillis + " мс");  
 }  
}

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 |

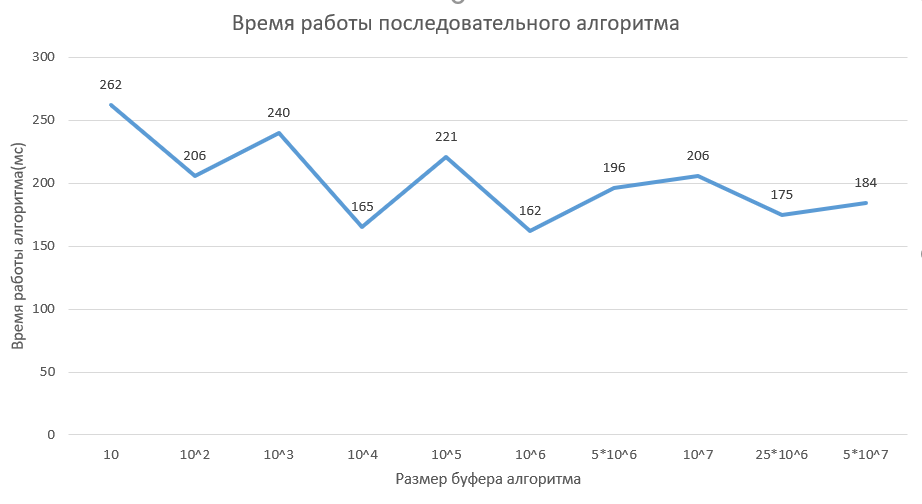
В ходе выполнения лабораторной работы были проведены измерения времени работы последовательного и параллельного алгоритмов. Результаты представлены в таблице 2.

Таблица 2 – Результаты работы алгоритмов

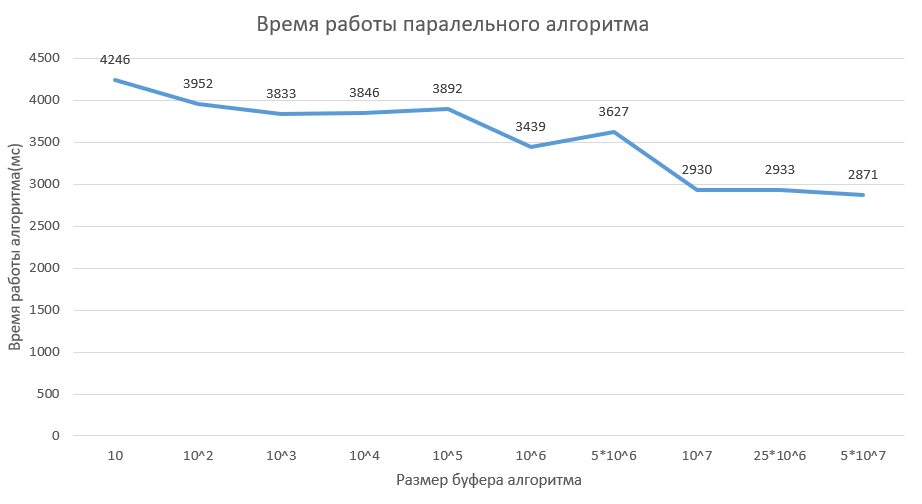
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Размер массива, эл | Размер буфера, эл | Время работы последовательного алгоритма , мс | Время работы параллельного алгоритма , мс | Ускорение |
| 108 | 10 | 262 | 4246 | 0.0617 |
| 108 | 102 | 206 | 3952 | 0.0521 |
| 108 | 103 | 240 | 3833 | 0.0626 |
| 108 | 104 | 165 | 3846 | 0.0429 |
| 108 | 105 | 221 | 3892 | 0.0568 |
| 108 | 106 | 162 | 3439 | 0.0471 |
| 108 | 5⋅106 | 196 | 3627 | 0.054 |
| 108 | 107 | 206 | 2930 | 0.0703 |
| 108 | 25⋅106 | 175 | 2933 | 0.0596 |
| 108 | 50⋅106 | 184 | 2871 | 0.064 |

Полученные данные можно представить в виде графиков.

На рисунке 1 представлен график зависимости времени работы последовательного алгоритма от изменения размера буфера при фиксированном размере массива.

  
Рисунок 1 – Зависимость времени работы последовательного алгоритма от размера буфера

На рисунке 2 представлен график зависимости времени работы параллельного алгоритма от изменения размера буфера при фиксированном размере массива.

  
Рисунок 2 – Зависимость времени работы параллельного алгоритма от размера буфера

На рисунке 3 представлен график ускорения работы программы.

Рисунок 3 – Ускорение работы программы

1. Заключение

Тестирование происходило с фиксированным размером массива = 108, а размер буфера менялся от небольшого значения = 10 до половины исходного массива = 5 \* 107.

Согласно полученным результатам, можно сделать вывод, что с увеличением размера буфера продолжительность работы последовательного алгоритма практически не изменяется, в отличие от продолжительности работы параллельного алгоритма [3].

Последовательный алгоритм не использует элементов синхронизации, в связи с чем он работает быстрее параллельного аналога.

Продолжительность работы параллельного алгоритма уменьшается с увеличением буфера в связи с тем, что уменьшается время простоя программы, связанное с синхронизацией. Действительно, если размер буфера мал, то каждый из потоков будет чаще простаивать, ожидая, пока другой поток выполнит свой набор инструкций [4].

Данный тип задачи не подходит для того, чтобы эффективно выполнить её, используя параллельный алгоритм. В результате, как и показал проведённый опыт, последовательная версия алгоритма работает намного быстрее.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Многопоточность в Java/Хабр. [Электронный ресурс] URL: https://habr.com/ru/post/164487/ (дата обращения: 30.09.2021).
2. Воеводин В.В., Воеводин Вл.В. Параллельные вычисления СПб.:БХВ-Петербург, 2002. - 608 с.
3. Грегори Р. Эндрюс Основы многопоточного, параллельного и распределенного программирования - Вильямс, 2002. - 512 с.
4. Воеводин В.В., Вычислительная математика и структура алгоритмов, ГСП-2, Москва, НИВЦ МГУ, 2006. – 113 c.

ПРИЛОЖЕНИЕ А

Код параллельной программы

import java.util.Arrays;  
import java.util.Random;  
import java.util.Scanner;  
  
class Producer implements Runnable {  
 int[] arr1;  
 Buffer buffer;  
 public Producer(int[] a, Buffer b) {  
 arr1 = a;  
 buffer = b;  
 }  
  
 @Override  
 public void run() {  
 for (int i = 0; i < arr1.length; ) {  
 if (buffer.Put(arr1[i])) {  
  
 i++;  
 }  
  
 }  
 }  
}  
  
class Consumer implements Runnable {  
 Buffer buffer;  
 int[] arr2;  
  
 public Consumer(int[] a, Buffer b) {  
 arr2 = a;  
 buffer = b;  
 }  
  
 @Override  
 public void run() {  
 int[] tmp = new int[1];  
 for (int i = 0; i < arr2.length; ) {  
 if (buffer.Read(tmp)) {  
 arr2[i] = tmp[0];  
  
 i++;  
 }  
  
 }  
 }  
}  
  
class Buffer {  
 int[] buffer1;  
  
 public Buffer(int buffer\_length) {  
 buffer1 = new int[buffer\_length];  
 }  
  
 @Override  
 public String toString() {  
 return "Buffer{" +  
 "buffer1=" + Arrays.*toString*(buffer1) +  
 '}';  
 }  
  
 int[] chet1 = new int[]{0};  
 int[] chet2 = new int[]{0};  
  
 public boolean Put(int a) {  
 if ((chet2[0] + 1) % buffer1.length != (chet1[0])) {  
 buffer1[chet2[0]] = a;  
  
 chet2[0] = (chet2[0] + 1) % buffer1.length;  
  
  
 return true;  
 } else  
 return false;  
 }  
  
 public boolean Read(int[] result) {  
 if (chet1[0] != chet2[0]) {  
 int b = chet1[0];  
  
 chet1[0] = (chet1[0] + 1) % buffer1.length;  
  
 result[0] = buffer1[b];  
 return true;  
 }  
 return false;  
 }  
}  
  
public class Main {  
 public static void main(String[] args) throws InterruptedException {  
 System.*out*.println("Лабораторная работа номер 1. Выполнил: Артамонов Владислав 6132");  
 int massive\_length = 100\_000\_000;  
 Scanner scan = new Scanner(System.*in*);  
 System.*out*.println("Длина массива 10^8 элементов.");  
 System.*out*.print("Введите длину буфера: ");  
 int buffer\_length = scan.nextInt();  
 int[] arr1 = new int[massive\_length];  
 int[] arr2 = new int[massive\_length];  
 Buffer buffer = new Buffer(buffer\_length);  
 long totalTime = 0;  
 int cycles = 5;  
 Random random = new Random(System.*currentTimeMillis*());  
 for (int j = 0; j < cycles; j++) {  
  
  
 for (int i = 0; i < massive\_length; i++) {  
 arr1[i] = random.nextInt();  
 }  
  
  
 Producer producer = new Producer(arr1, buffer);  
 Consumer consumer = new Consumer(arr2, buffer);  
 Thread producer\_thread = new Thread(producer);  
 Thread consumer\_thread = new Thread(consumer);  
  
 long start = System.*currentTimeMillis*();  
 producer\_thread.start();  
 consumer\_thread.start();  
 consumer\_thread.join();  
 long finish = System.*currentTimeMillis*();  
  
  
 long timeConsumedMillis = (finish - start) /\*/ cycles\*/;  
 totalTime += timeConsumedMillis;  
 System.*out*.println(j + ") Время работы параллельной программы: " + timeConsumedMillis + " мс");  
 System.*out*.println(Arrays.*equals*(arr1, arr2));  
 }  
 System.*out*.println("Среднее время работы параллельной программы: " + totalTime / cycles + " мс");  
 }  
  
}

ПРИЛОЖЕНИЕ Б

Код последовательной программы

import java.util.Scanner;  
  
public class posled {  
 public static void main(String[] args) {  
 System.*out*.println("Лабораторная работа номер 1. Выполнил: Артамонов Владислав 6132");  
 int massive\_length = 100\_000\_000;  
 Scanner scan = new Scanner(System.*in*);   
 System.*out*.println("Длина массива - 10^8 элементов.");  
 System.*out*.print("Введите размер буфера: ");  
 int buffer\_length = scan.nextInt();  
 int[] arr1 = new int[massive\_length];  
 for (int i = 0; i < massive\_length; i++) {  
 arr1[i] = i;  
 }  
 int[] arr2 = new int[massive\_length];  
 int[] buffer = new int[buffer\_length];  
 int pointer = 0;  
 int counter\_cycles = 5;  
 long start = System.*currentTimeMillis*();  
 for (int i = 0; i < counter\_cycles; i++) {  
 while (pointer + buffer\_length <= massive\_length) {  
 for (int j = 0; j < buffer\_length; j++) {

buffer[j] = arr1[j + pointer];

}

for ( int j = 0; j < buffer\_length; j++) {

arr2[j + pointer] = buffer[j];

}  
 pointer = pointer + buffer\_length;  
 }  
 pointer = 0;  
 }  
 long finish = System.*currentTimeMillis*();  
 long timeConsumedMillis = (finish - start) / counter\_cycles;  
 System.*out*.println("Время работы : " + timeConsumedMillis + " мс");  
 }  
}