Федеральное государственное бюджетное образовательное учреждение

высшего образования

«Саратовский государственный технический университет

имени Гагарина Ю.А.»

Институт электронной техники и приборостроения

Кафедра Информационная безопасность автоматизированных систем

Направление 10.05.03 Информационная безопасность автоматизированных систем

**Практическая работа №5**

по дисциплине «Параллельные системы и их программирование»

по теме: «Реализация последовательного и параллельного алгоритмов метода Гаусса»

|  |  |
| --- | --- |
|  | Выполнил: студент 4 курса  учебной группы с-ИБС42  очной формы обучения  Солодилов В.В.  Проверил: профессор. каф. ИБС  Кондратов Д.В. |

Саратов 2022

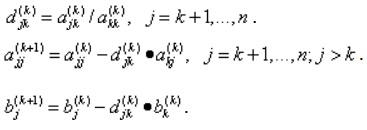
**Задание**

Реализуйте программу для решения системы линейных алгебраических уравнений (СЛАУ) методом Гаусса, используя последовательный и параллельный алгоритмы решения задачи. При реализации параллельного алгоритма используйте различные средства синхронизации. Сделайте вывод об эффективности применения параллельного алгоритма.

**Практическая часть**

Для реализации последовательного алгоритма решения задачи методом Гаусса были произведены следующие действия:

1. Матрицы коэффициентов при x и свободных членов были заполнены с помощью функции Math.random() \* 10, т.е. случайными значениями в промежутке от 0 до 10.
2. Полученные значения матриц были использованы в прямом ходе для нахождения приведённой матрицы, причём её значения найдены с использованием различных преобразований, которые можно выразить с помощью следующих формул:

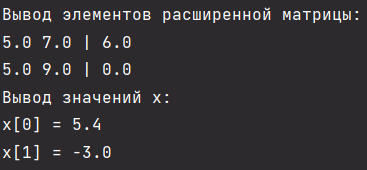


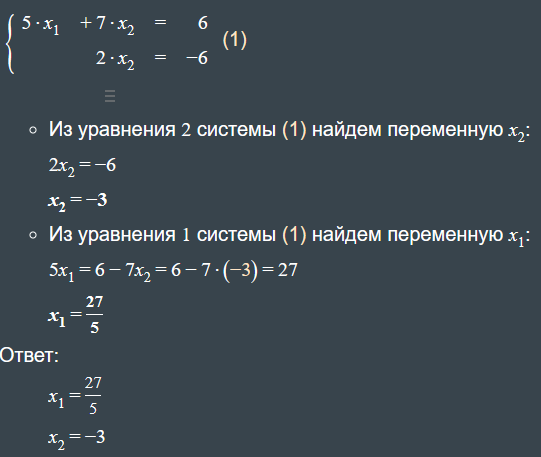
1. Приведённая матрица использовалась в обратном ходе, суть которого заключается в подстановке последующего значения x в предыдущее до тех пор, пока все значения x не будут найдены.
2. Найденные значения x и время выполнения программы были выведены на экран.

При распараллеливании последовательного алгоритма использовались Lock’и, атомарные операторы и метод ожидания завершения потока join(), причём для прямого метода использовались первые два средства синхронизации, а для обратного – последний. Полученные значения x также были выведены на экран.

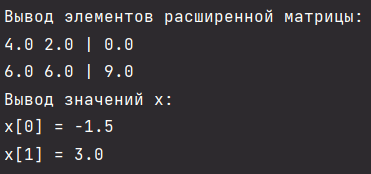
Осуществим проверку правильности реализации алгоритмов. Для этого решим системы СЛАУ сначала программно, затем с помощью интернет-источника и сравним полученные значения x.

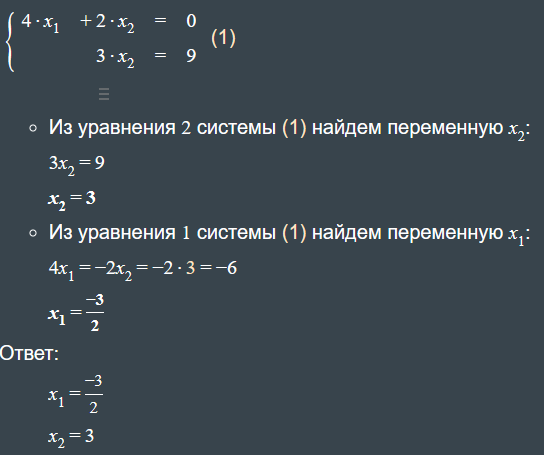
Для последовательного алгоритма имеем:





В результате выполнения программы, реализующей параллельный алгоритм, получаем:





Значения, полученные при решении СЛАУ методом Гаусса для последовательного и параллельного алгоритмов, совпадают, следовательно, алгоритмы решения задачи реализованы правильно.

**Вывод**

В результате выполнения практической работы были реализованы последовательный и параллельный алгоритмы решения задачи методом Гаусса на языке программирования Java. По итогам выполнения программы можно сказать, что при малом порядке СЛАУ последовательный алгоритм будет эффективнее из-за меньших затрат по сравнению с затратами на организацию многопоточности, однако при порядка системы > 100 параллельный алгоритм становится более эффективным по сравнению с последовательным.

**Листинг программы**

//Файл Actions.java

public class *Actions* {  
 static void directMethodFunction(double[][] *startMatrix*, double[] *freeMatrix*, int *k*) {  
 for (int j = *k* + 1; j < *startMatrix*.length; j++) {  
 double d = *startMatrix*[j][*k*] / *startMatrix*[*k*][*k*];  
 for (int i = *k*; i < *startMatrix*.length; i++) {  
 *startMatrix*[j][i] = *startMatrix*[j][i] - d \* *startMatrix*[*k*][i];  
 }  
 *freeMatrix*[j] = *freeMatrix*[j] - d \* *freeMatrix*[*k*];  
 }  
 }  
 public static double[] xCalculation(double[][] *matrix*, double[] *freeMatrix*) {  
 double d, s;  
 double[] x = new double[*matrix*.length];  
 for (int k = *matrix*.length - 1; k >= 0; k--) {  
 d = 0;  
 for (int j = k + 1; j < *matrix*.length; j++) {  
 s = *matrix*[k][j] \* x[j];  
 d = d + s;  
 }  
 x[k] = (*freeMatrix*[k] - d) / *matrix*[k][k];  
 }  
 return x;  
 }  
  
 static void createStartMatrices(int *length*, double[][] *startMatrix*, double[] *freeMatrix*) {  
 for (int i = 0; i < *length*; i++) {  
 *freeMatrix*[i] = (int) (*Math*.*random*() \* 10);  
 for (int j = 0; j < *length*; j++) {  
 *startMatrix*[i][j] = (int) (*Math*.*random*() \* 10);  
 }  
 }  
 }  
  
 static void printMatrix(double[][] *startMatrix*, double[] *freeMatrix*) {  
 *System*.out.println("Вывод элементов расширенной матрицы:");  
 for (int i = 0; i < *startMatrix*.length; i++) {  
 for (int j = 0; j < *startMatrix*.length; j++) {  
 *System*.out.print(*startMatrix*[i][j] + " ");  
 }  
 *System*.out.println("| " + *freeMatrix*[i]);  
 }  
 }  
  
 static void printX(double[] *x*) {  
 *System*.out.println("Вывод значений x:");  
 for (int i = 0; i < *x*.length; i++) {  
 *System*.out.println("x[" + i + "] = " + *x*[i]);  
 }  
 }  
}

//Файл NoParallelMain.java

import *java.util.Scanner*;  
public class *NoParallelMain* extends *Actions* {  
 private static final *Scanner* scanner = new Scanner(*System*.in);  
 record *MatrixResult*(double[][] endMatrix, double[] freeMatrix) {}  
 public static *MatrixResult* matrixTransformation(double[][] *matrix*, double[] *freeMatrix*) {  
 for (int k = 0; k < *matrix*.length; k++) {  
 *directMethodFunction*(*matrix*, *freeMatrix*, k);  
 }  
 return new *NoParallelMain*.MatrixResult(*matrix*, *freeMatrix*);  
 }  
  
 public static void main(*String*[] *args*) {  
 *System*.out.println("Решение СЛАУ методом Гаусса без распараллеливания вычислений...");  
 *System*.out.print("Введите порядок СЛАУ: ");  
 int length = scanner.nextInt();  
  
 double[][] startMatrix = new double[length][length];  
 double[] freeMatrix = new double[length];  
 double[] x;  
 long time = *System*.*currentTimeMillis*();  
  
 *createStartMatrices*(length, startMatrix, freeMatrix);  
 *printMatrix*(startMatrix, freeMatrix);  
  
 *MatrixResult* matrixResult = *matrixTransformation*(startMatrix, freeMatrix);  
 x = *xCalculation*(matrixResult.endMatrix(), matrixResult.freeMatrix());  
 *printX*(x);  
  
 *System*.out.println("Время, затраченное на выполнение программы - " + (*System*.*currentTimeMillis*() - time) + " мс");  
 }  
}

//Файл ParallelMain.java

import *java.util.Scanner*;  
import *java.util.concurrent.TimeUnit*;  
import *java.util.concurrent.atomic.AtomicInteger*;  
import *java.util.concurrent.locks.Lock*;  
import *java.util.concurrent.locks.ReentrantLock*;  
  
public class *ParallelMain* extends *Actions* {  
 private static final *Scanner* scanner = new Scanner(*System*.in);  
  
 public static void main(*String*[] *args*) throws *InterruptedException* {  
 *System*.out.println("Решение СЛАУ методом Гаусса с распараллеливанием вычислений...");  
 *System*.out.print("Введите порядок СЛАУ: ");  
 int length = scanner.nextInt();  
  
 *AtomicInteger* directMethodValue = new AtomicInteger(0);  
 *AtomicInteger* reverseMethodValue = new AtomicInteger(length - 1);  
 *Lock* lock = new ReentrantLock();  
  
 int cores = *Runtime*.*getRuntime*().availableProcessors();  
 double[][] startMatrix = new double[length][length];  
 double[] freeMatrix = new double[length];  
 double[] x = new double[length];  
 double n = *Math*.*ceil*((double) length / cores);  
  
 *createStartMatrices*(length, startMatrix, freeMatrix);  
 long time = *System*.*currentTimeMillis*();  
  
 *Runnable* directMethod = () -> {  
 try {  
 if (lock.tryLock(10, *TimeUnit*.MILLISECONDS)) {  
 for (int k = directMethodValue.get(); k < directMethodValue.get() + n; k++) {  
 if (k >= length) {  
 break;  
 } else {  
 *directMethodFunction*(startMatrix, freeMatrix, k);  
 }  
 directMethodValue.incrementAndGet();  
 }  
 }  
 } catch (*InterruptedException e*) {  
 throw new RuntimeException(*e*);  
 } finally {  
 lock.unlock();  
 }  
 };  
  
 *Runnable* reverseMethod = () -> {  
 try {  
 if (lock.tryLock(10, *TimeUnit*.MILLISECONDS)) {  
 for (int k = reverseMethodValue.get(); k >= reverseMethodValue.get() - n; k--) {  
 if (reverseMethodValue.get() < 0) {  
 break;  
 } else {  
 double d = 0;  
 for (int j = k + 1; j < startMatrix.length; j++) {  
 double s = startMatrix[k][j] \* x[j];  
 d += s;  
 }  
 x[k] = (freeMatrix[k] - d) / startMatrix[k][k];  
 reverseMethodValue.decrementAndGet();  
 }  
 }  
 }  
 } catch (*InterruptedException e*) {  
 throw new RuntimeException(*e*);  
 } finally {  
 lock.unlock();  
 }  
 };  
  
 *createStartMatrices*(length, startMatrix, freeMatrix);  
  
 *printMatrix*(startMatrix, freeMatrix);  
  
 *Thread*[] threadsDirectMethod = new Thread[cores];  
 for (int i = 0; i < threadsDirectMethod.length; i++) {  
 threadsDirectMethod[i] = new Thread(directMethod);  
 threadsDirectMethod[i].start();  
 threadsDirectMethod[i].join();  
 }  
  
 *Thread*[] threadsReverseMethod = new Thread[cores];  
 for (int i = 0; i < threadsDirectMethod.length; i++) {  
 threadsReverseMethod[i] = new Thread(reverseMethod);  
 threadsReverseMethod[i].start();  
 threadsReverseMethod[i].join();  
 }  
  
 *printX*(x);  
  
 *System*.out.println("Время, затраченное на выполнение программы - " + (*System*.*currentTimeMillis*() - time) + " мс");  
  
 *System*.*exit*(0);  
 }  
}