МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІСЬКА ПОЛІТЕХНІКА»

Кафедра ІСМ



**Звіт**

До розрахунково-графічної роботи

З дисципліни:

«Технології розподілених систем та паралельних обчислень»

Варіант №7

*Виконав:*

*студент групи КН-47*

Іванов Вадим

*Прийняв:*

*доцент* Василюк А. С.

Львів 2018

**ЗМІСТ**

1. Завдання …………………………………………………………………………..3
2. Вступ……………………………………………………………………………....4
3. Розділ 1. Теоретична частина…………………………………………………....5
4. Розділ 2. Практичні частини....………………………………………………….8
5. Висновки………………………………………………………………………….9
6. Список використаної літератури………………………………………………..10

**ЗАВДАННЯ**

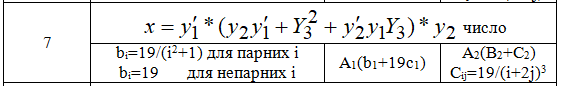
**Завдання 1. (теоретичне)**. У відповідності до варіанту (визначається згідно з номером студента в журналі викладача) розкрити теоретичні положення згідно з темою розрахункової роботи.

Варіант 7. Способи розпаралелювання – за керуванням та за інформацією.

**Завдання 2 (практичне).** Паралельні обчислення**.**

Використовуючи метод функціональної декомпозиції, розробити алгоритм обчислення запропонованого матрично-векторного виразу, який би враховував можливість паралельного виконання. ВСІ вхідні дані є цілими числами, більшими за нуль. ПРОВЕСТИ ВЕРИФІКАЦІЮ АЛГОРИТМУ.

На основі створеного алгоритму написати програму яка дозволяє обчислити вираз.



**Завдання 3 (практичне). Розподілені системи.**

В даному розділі необхідно реалізувати систему у відповідності до технологій побудови розподілених систем, а саме необхідно:

* сформулювати поставлену задачі, вимоги до програми, що розробляється.
* проаналізувати існуючі рішення поставленої задачі та описати її особливості
* обґрунтувати вибраний метод розв'язання задачі
* описати та відобразити алгоритм, за допомогою якого буде розв'язана задача (необхідно описати кожний крок алгоритму, вказати граничні умови, дати оцінку швидкості алгоритму та об'єму необхідної пам'яті)
* реалізувати програмне рішення (при описані програми необхідно зробити співставлення частин алгоритму та фрагментів програми)
* в контрольних прикладах необхідно показати 1-2 сеанси роботи з програмою, що демонструють основні можливості роботи програми.

Варіант 7. Розподілена система прийняття рішень в генерації моделей алгоритмів

**ВСТУП**

Для розв’язання багатьох задач (прогноз погоди, задачі гідро- і газодинаміки, квантової хімії, астрономії, спектроскопії, біології, ядерної фізики тощо) необхідна висока продуктивність обчислень, висока швидкість передачі інформації по каналах зв’язку та великі об’єми оперативної і постійної пам’яті. Одним з шляхів забезпечення таких вимог є організація паралельних обчислювальних процесів і відповідних технічних засобів їх реалізації. Причому, ефективність паралельної обробки залежить як від продуктивності комп’ютерів, так і від розмірів і структури пам’яті, пропускної здатності каналів зв’язку, використаних мов програмування, компіляторів, операційних систем, чисельних методів та інших математичних досліджень. Такий широкий обсяг параметрів вимагає проведення досліджень на різних рівнях: на рівні розпаралелення алгоритмів, створення спеціальних мов програмування, компіляторів, багатопроцесорних систем, неоднорідних систем, кластерів.

Для скорочення термінів розробки паралельних засобів та дослідження їх роботи використовується моделювання.

Метою виконання роботи є засвоєння основних методів та алгоритмів моделювання паралельних обчислювальних процесів, принципів побудови відповідних структур, набуття початкових практичних навиків.

**РОЗДІЛ 1. ТЕОРЕТИЧНА ЧАСТИНА**

Можливі наступні режими виконання незалежних частин програми:

* Паралельне виконання - в один і той же момент часу виконується кілька команд обробки даних; цей режим обчислень може бути забезпечений не тільки наявністю декількох процесорів, але і за допомогою конвеєрних і векторних обробних пристроїв;
* Розподілені обчислення - цей термін зазвичай застосовують для вказівки способу паралельної обробки даних, при якій використовуються декілька обробних пристроїв, досить віддалених один від одного і в яких передача даних по лініях зв'язку призводить до суттєвих тимчасових затримок.

При такому способі організації обчислень ефективна обробка даних тільки для паралельних алгоритмів з низькою інтенсивністю потоків міжпроцесорних передач даних; таким чином функціонують, наприклад, багатомашинні обчислювальні комплекси, утворені об'єднанням декількох окремих електронно-обчислювальних машин за допомогою каналів зв'язку локальних або глобальних інформаційних мереж. Формально до цього списку може бути віднесений і багатозадачність (режим поділу часу), при якому для виконання процесів використовується єдиний процесор; в цьому режимі зручно відлагоджувати паралельні додатки.

Існує два способи паралельної обробки даних: паралелізм і конвеєрні. Паралелізм припускає наявність p однакових пристроїв для обробки даних і алгоритм, що дозволяє виробляти на кожному незалежну частину обчислень, в кінці обробки часткові дані збираються разом для отримання остаточного результату. У цьому випадку отримаємо прискорення процесу в p разів. Далеко не кожен алгоритм може бути успішно розпаралелений таким чином (природною умовою розпаралелювання є обчислення незалежних частин вихідних даних по однаковим - чи подібним - процедурам; ітераційного або рекурсивність викликають найбільші проблеми при розпаралелювання).

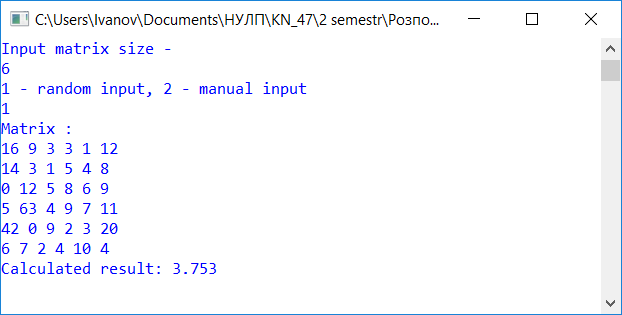
Ідея конвеєрної обробки полягає у виділенні окремих етапів виконання загальної операції, причому кожен етап після виконання своєї роботи передає результат наступного, одночасно приймаючи нову порцію вхідних даних. Кожен етап обробки виконується своєю частиною пристрою обробки даних (щаблем конвеєра), кожна щабель виконує певну дію (мікрооперацію); загальна обробка даних вимагає спрацьовування цих частин послідовно. Конвеєрна при виконанні команд імітує роботу конвеєра складального заводу, на якому виріб послідовно проходить ряд робочих місць; причому на кожному з них над виробом виробляється нова операція. Ефект прискорення досягається за рахунок одночасної обробки ряду виробів на різних робочих місцях.

Прискорення обчислень досягається за рахунок використання всіх ступенів конвеєра для потокової обробки даних (дані потоком надходять на вхід конвеєра і послідовно обробляються на всіх щаблях). Конвеєри можуть бути скалярними або векторними пристроями (різниця полягає лише в тому, що в останньому випадку можуть бути використані обробні вектори команди). У разі довжини конвеєра l час обробки n незалежних операцій складе l + n-1 (кожен ступінь спрацьовує за одиницю часу). При використанні такого пристрою для обробки єдиною порції вхідних даних буде потрібно час l × n і тільки для безлічі порцій отримаємо прискорення обчислень, близьке до l. Кожен з цих двох підходів має свої переваги і недоліки. Композиційна схема потребує складної і дорогої апаратури композиції, що обмежує кількість можливих атрибутів у пікселі. Крім того, навантаження на рендери може бути не збалансованим. З іншого боку, схема розподілу екрана припускає наявність перерозподільчої сітки, яка дозволяє довільним чином розподілити полігони для растеризації в різних частинах екрану, що надходять від геометричних процесорів. Навпаки, асемблування різних ділянок в повне зображення доволі легко, так як не потребує виконання операції композиції пікселів. Логічний підрозділ елементів моделі сцени реалізує третій підхід до розпаралелення – структурного. Кожен об'єкт сцени, що має свою специфіку (наприклад, форма подання) позначається і кожен процесор у структурі обробки реагує на запити тільки об'єктів свого класу. Такий підхід ефективний, зокрема, при одночасному моделюванні природних і штучних об'єктів. Це зв'язано з використанням різних способів візуалізації. Конкретна технічна реалізація підсистеми растрування визначається базовими алгоритмами візуалізації і необхідними характеристиками продуктивності. Найбільшого поширення отримав просторовий підхід до розпаралелення процесу формування зображення в реальному масштабі часу, де просторовий паралелізм підтримується конвеєрною архітектурою. Розрізняють такі основні рівні паралелізму, які використовуються при розпаралеленні різних етапів формування реалістичних зображень (розпаралелення на нижньому рівні): 1) рівень ЕЗ (пікселя), де може забезпечуватися паралельне обчислення по шести координатах (х, у, z, R, G, B) і доступ до кадрового буфера; 2) рівень групи ЕЗ, де значення інтенсивностей і координат обчислюються паралельно для декількох ЕЗ (зокрема, цей тип паралелізму реалізується в описаних структурах трасування променів); 3) рівень примітива, де паралелізм реалізується, наприклад, усередині трикутника, забезпечуючи паралельне обчислення для його горизонтальних і вертикальних лінійних або прямокутних фрагментів; 4) рівень групи примітивів, де паралелізм забезпечується одночасним раструванням ряду примітивів; 5) рівень кадру, де паралелізм досягається одночасною обробкою ряду підкадрів чи повних кадрів.

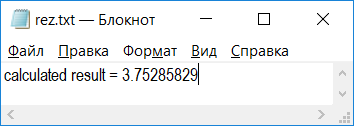
**РОЗДІЛ 2.**

**ПРАКТИЧНА ЧАСТИНА 1**

У даній практичні частині було створено сайт аплікацію яка використовує метод функціональної декомпозиції та реалізує алгоритм матрично-векторного виразу, який би враховував можливість паралельного виконання. Також було проведено верифікацію алгоритму. Аплікація створення за допомогою об’єктно орієнтованої мови програмування C#.



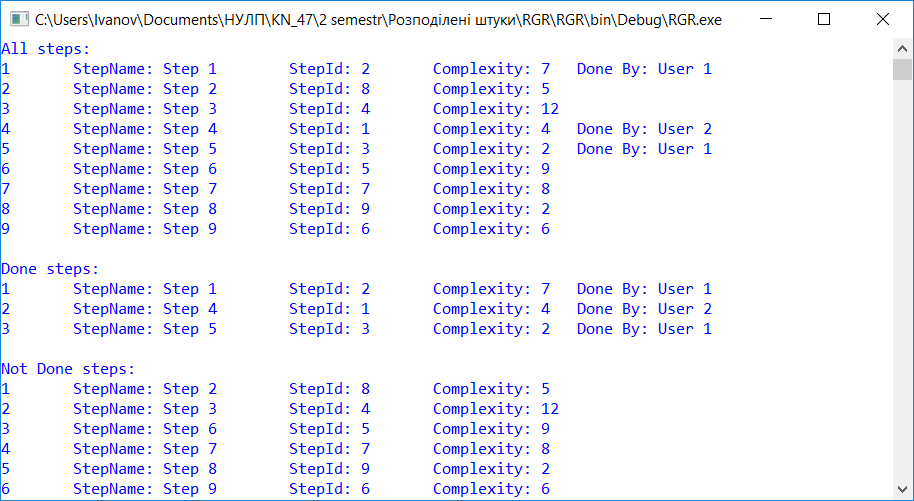
*Рис. 1. Результат виконання завдання.*



*Рис. 2. Результат виконання завдання, записаний у файл.*

**ПРАКТИЧНА ЧАСТИНА 2**

У даній практичні частині створюється аплікація що реалізує розподілену систему прийняття рішень в генерації моделей алгоритмів, що допомагає зрозуміти, які кроки алгоритму були зроблені та яким користувачем. Аплікація створення за допомогою об’єктно орієнтованої мови програмування C#.



*Рис. 2. Список кроків алгоритму.*

**ВИСНОВОК :** Виконання теоретичної частино даної розрахунково-графічної роботи дало змогу ознайомитися зі способами розпаралелювання – за керуванням та за інформацією. У першій практичні частині було створено консольну аплікацію яка використовує метод функціональної декомпозиції та реалізує алгоритм матрично-векторного виразу, який би враховував можливість паралельного виконання. Також було проведено верифікацію алгоритму. У другій практичні частині було створено консольну аплікацію що реалізує розподілену систему опліку купівлі-продажу замельних ділянок яка дозволяє переглядати куплені, не куплені та всі земельні ділянки.

**СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ**

1. Документація з використання ASP.NET Web API 2 (C#) [Електронний ресурс]. Режим доступу URL: https://docs.microsoft.com/en-us/aspnet/web-api/overview/getting-started-with-aspnet-web-api/tutorial-your-first-web-api.
2. Офіційна документація Bootstrap. [Електронний ресурс]. Режим доступу URL: https://getbootstrap.com/.
3. Форум програмістів Електронний ресурс]. Режим доступу URL: https://stackoverflow.com/.
4. ГОСТ 34.201-90. Информационная технология. Комплекс стандартов и руководящих документов на автоматизированные системы. Автоматизированные системы**.** Стадии создания. – М.: Изд-во стандартов, 1991. – 16 с.
5. ДСТУ 3008-95. Документація. Звіти у сфері науки і техніки. Структура і правила оформлення. – К.: Держстандарт України, 1995. – 36 с.
6. Сторінка в ресурсі Вікіпедія. [Електронний ресурс]. Режим доступу URL: https://uk.wikipedia.org/wiki/PHP

Додаток А

using System;

using System.Collections.Generic;

using System.Linq;

using System.Runtime.InteropServices;

namespace RGP

{

class LandPlot

{

public string Name { get; set; }

public double Price { get; set; }

public double Size { get; set; }

public bool Bought { get; set; }

public string BoughtName { get; set; }

}

class Program

{

static void Main(string[] args)

{

var algoritm = new List<LandPlot>() = AutoGenData()

Console.WriteLine("All steps:");

algoritm.ForEach(i => {

Show(algoritm.IndexOf(i), i);

});

Console.WriteLine("\nDone steps:");

var donealgoritm = algoritm.Where(i => i.Dobe).ToList();

donealgoritm.ForEach(i => {

Show(donealgoritm.IndexOf(i), i);

});

Console.WriteLine("\nNot Done steps:");

var notDonealgoritm = algoritm.Where(i => !i.Done).ToList();

notDonealgoritm.ForEach(i => {

Show(notDonealgoritm.IndexOf(i), i);

});

Console.WriteLine();

Console.WriteLine("Input matrix size - ");

Console.ReadLine();

Console.WriteLine("1 - random input, 2 - manual input");

Console.ReadLine();

if(I = 1)

var matrix = random();

Console.WriteLine("Matrix : ");

for (int i = 0; i < 6; i++)

{

for (int j = 0; j < 6; j++)

{

Console.Write($"{matrix[i,j]} ");

}

Console.WriteLine();

}

Console.WriteLine("Calculated Result : " + Decocmposition());

File file = new file("rez.txt ");

File.Write("calculated result = " + Decocmposition());

Console.WriteLine();

Console.ReadKey();

}

static void Show(int index, LandPlot item)

{

Console.Write("{0}", index + 1);

Console.Write("\tStepName: " + item.Name);

Console.Write("\tStepId: " + item.Size);

Console.Write("\tСomplexity: " + item.Price);

if (item.Bought)

{

Console.Write("\tDone By: " + item.BoughtName);

}

Console.WriteLine();

}

}

}

class Decomposition {

private double i { get; set; }

private double j { get; set; }

private double b1 { get { return 7; } }

private double b2 { get { return 1 / Math.Pow( i, 3 ); } }

private double C { get { return 1 / Math.Pow( i + j, 3 ); } }

public double CalculateX( Matrix m ) {

i = 1;

j = 1;

return C;

}

}

public class Matrix {

public Matrix( int [,] array ) {

this.array = array;

}

private int [,] array;

public int Height {

get {

return array.GetLength( 0 );

}

}

public int Width {

get {

return array.GetLength( 1 );

}

}

public static double GetComplementaryMinor( Matrix M, int k ) {

return GetMinor( M, k - 1 );

}

public static double GetMinor( Matrix matrix, int k ) {

var maxMinorRange = Math.Min( matrix.Height, matrix.Width );

var square = new int [maxMinorRange, maxMinorRange];

//Getting square from rectengle

for (int i = 0; i < maxMinorRange; i++) {

for (int j = 0; j < maxMinorRange; j++) {

square [i, j] = matrix.array [i, j];

}

}

return GetMinor( new Matrix( square ), k, k );

}

public static double GetMinor( Matrix matrix, int i, int j ) {

//In case non-square matrix

if (matrix.Height != matrix.Width) {

throw new ArgumentException( "No i- j- mionors for rectangle matrix" );

}

var size = matrix.Width;

//Matrix with removing row i

var matrixWithoutRow = new int [size - 1, size];

//Result matrix - matrix with removing row i and collum j

var resultMatrix = new int [size - 1, size - 1];

//Creating Matrix with removing row i

for (int n = 0; n < size - 1; n++) {

for (int m = 0; m < size; m++) {

if (n < i) {

matrixWithoutRow [n, m] = matrix.array [n, m];

} else {

matrixWithoutRow [n, m] = matrix.array [n + 1, m];

}

}

}

//Creating result matrix

for (int n = 0; n < size - 1; n++) {

for (int m = 0; m < size - 1; m++) {

if (m < j) {

resultMatrix [n, m] = matrixWithoutRow [n, m];

} else {

resultMatrix [n, m] = matrixWithoutRow [n, m + 1];

}

}

}

return Determinant( new Matrix( resultMatrix ) );

}

public static double AddMinors( double minor1, double minor2 ) {

return minor1 + minor2;

}

public static double MultiplicateMinors( double minor1, double minor2 ) {

return minor1 \* minor2;

}

public static double DivideMinors( double minor1, double minor2 ) {

return minor1 / minor2;

}

public static double Determinant( Matrix inputSquare ) {

//In case of one-value matrix.

if (inputSquare.Height == 1) {

return inputSquare.array [0, 0];

}

//In case of two-order matrix.

else if (inputSquare.Height == 2) {

//Calculating by mathematical formula.

return inputSquare.array [0, 0] \* inputSquare.array [1, 1]

- inputSquare.array [1, 0] \* inputSquare.array [0, 1];

}

double result = 0;

int size = inputSquare.Width;

for (int i = 0; i < size; i++) {

//Calculating by mathematical formula.

result += Math.Pow( -1, i ) \* inputSquare.array [0, i] \* GetMinor( inputSquare, 0, i );

}

return result;

}

public override string ToString() {

StringBuilder result = new StringBuilder();

for (int i = 0; i < Height; i++) {

result.Append( "( " );

for (int j = 0; j < Width; j++) {

result.Append( array [i, j] );

result.Append( " " );

}

result.Append( ")" );

result.AppendLine();

}

return result.ToString();

}

}

}