Міністерство освіти і науки України

Національний університет „Львівська політехніка”

Кафедра ЕОМ



**Звіт**

з лабораторної роботи №4

з дисципліни: “Паралельні та розподілені обчислення”

на тему: “Паралельні алгоритми множення матриці на вектор”

Виконав: ст. гр. КІ-33

Лобай Р.І.

Прийняв: викладач

Козак Н.Б.

Львів – 2020

**Мета:** ознайомитись з методами організації паралельного множення матриці на вектор та розробити паралельну програму з використанням технології MPI.

**Індивідуальне завдання:** для заданої у варіанті кількості процесорів розробити програму для паралельного перемноження матриці на вектор заданого розміру з використанням МРІ.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **№ варіанту** | **Розмір матриці** | | **Тип розбиття** | **Кількість процесорів** |
| 9 | 250 | 750 | блокове | 25 |

**Хід роботи:**

**Підготовка**

**Розбиття матриці**

При блоковому способі розбиття даних (розбиття матриці на блоки однакового розміру) вхідна матриця буде мати такий вигляд:

750

250

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

. . . . . . . . . . . . . . . . . .

30

*Рис.1. Розбиття вхідної матриці на блоки для 25 процесорів*

Для кожного процесора визначено наступний розмір блоку для таких параметрів: матриця *А* розмірності *m х n*, вектор *b*, що складається з *n* елементів та вектора результатів *с* розміру *m.* Вважається, що вектори *b* і *c* копіюються на кожний процесор.

Тоді: m x n / p + n + m = 250 x 750 / 25 + 250 + 750 = 8 500 елементів;

Кількість операцій визначається на основі формули:

*T1 = m·(2n-1) = 250 \* 1 499*

та становить 374 750 операцій для кожного процесора.

**Розробка схеми інформаційної взаємодії**

Для виконання базової підзадачі скалярного добутоку процесор повинен містити відповідний блок матриці *А* і копію вектора *b*. Після завершення обчислень кожна базова підзадача визначає один з елементів вектора результату *с*.

Для об'єднання результатів і отримання повного вектора *с* на кожному з процесорів обчислювальної системи необхідно виконати операцію узагальненого збору даних, в якій кожен процесор передає свій обчислений елемент вектора *с* решті всіх процесорів за допомогою функції *MPI\_gather* з бібліотеки MPI.

У загальному вигляді схема інформаційної взаємодії підзадач в ході виконуваних обчислень наведена на рис. 3.



*Рис. 2. Організація обчислень при виконанні паралельного алгоритму множення матриці на вектор на основі розбиття матриці по блоках*

**Розробка програми з використанням МРІ**

Програма реалізує логіку роботи алгоритму, послідовно викликає необхідні підпрограми, написана на мові програмування C# із використання бібліотеки MPI.NET

using System;

using System.IO;

using MPI;

namespace lab4

{

class Program

{

static string FileNameMatrix = "D:\\Matrix.txt";

static string FileNameVector = "D:\\Vector.txt";

static string FileNameResult = "D:\\Result.txt";

static int MatrixHeight = 250;

static int MatrixLenght = 750;

static int ProcessorCount = 25;

static void Main(string[] args)

{

using (new MPI.Environment(ref args))

{

int[,] Matrix;

int[] Vector;

int[] Result;

RandomGenericValue();

Matrix = new int[MatrixHeight, MatrixLenght];

Vector = new int[MatrixLenght];

Result = new int[MatrixHeight];

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

{

Result[i] = 0;

}

int[] CheckResult = new int[MatrixHeight];

int[] StartX = new int[ProcessorCount];

int[] EndX = new int[ProcessorCount];

int[] StartY = new int[ProcessorCount];

int[] EndY = new int[ProcessorCount];

int BlockSizeLenght;

int BlockSizeHeight;

int BlockSizeX = (int)Math.Sqrt(ProcessorCount);

int BlockSizeY = ProcessorCount / BlockSizeX;

if (BlockSizeX \* BlockSizeY != ProcessorCount)

{

BlockSizeX = 2;

BlockSizeY = ProcessorCount / BlockSizeX;

}

int[] MatrixHeightProc = new int[ProcessorCount];

int[] MatrixLenghtProc = new int[ProcessorCount];

if (MatrixHeight % BlockSizeX == 0)

{

BlockSizeLenght = BlockSizeY;

BlockSizeHeight = BlockSizeX;

int CheckMatrixLenght = MatrixLenght % BlockSizeY \* BlockSizeX;

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

{

MatrixHeightProc[i] = MatrixHeight / BlockSizeX;

MatrixLenghtProc[i] = MatrixLenght / BlockSizeY;

}

for (int i = 0; i < ProcessorCount && CheckMatrixLenght > 0; i++)

{

CheckMatrixLenght--;

MatrixLenghtProc[i]++;

}

}

else

{

BlockSizeLenght = BlockSizeX;

BlockSizeHeight = BlockSizeY;

int CheckMatrixLenght = MatrixLenght % BlockSizeX \* BlockSizeY;

int CheckMatrixHeight = MatrixHeight % BlockSizeY \* BlockSizeX;

int CheckMatrixHeightH = CheckMatrixHeight / 2;

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

{

MatrixHeightProc[i] = MatrixHeight / BlockSizeY;

MatrixLenghtProc[i] = MatrixLenght / BlockSizeX;

}

for (int i = 0; i < ProcessorCount && CheckMatrixLenght > 0; i++)

{

CheckMatrixLenght--;

MatrixLenghtProc[i]++;

}

for (int i = 0; i < ProcessorCount && CheckMatrixHeight > 0; i++)

{

MatrixHeightProc[i]++;

CheckMatrixHeight--;

if (CheckMatrixHeightH == CheckMatrixHeight)

i = ProcessorCount / 2 - 1;

}

}

int NextX = 0;

int NextY = 0;

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

{

StartX[i] = NextX;

EndX[i] = StartX[i] + MatrixHeightProc[i] - 1;

NextX += MatrixHeightProc[i];

StartY[i] = NextY;

EndY[i] = StartY[i] + MatrixLenghtProc[i] - 1;

if (NextX >= MatrixHeight)

{

NextY += MatrixLenghtProc[i];

NextX = 0;

}

}

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

{

Console.WriteLine(MatrixHeightProc[i] + " " + MatrixLenghtProc[i]);

}

Console.WriteLine("\n\n");

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

{

Console.WriteLine(StartX[i] + " - " + EndX[i] + " " + StartY[i] + " - " + EndY[i]);

}

Intracommunicator comm = Communicator.world;

string MatrixText = File.ReadAllText(FileNameMatrix);

string VectorText = File.ReadAllText(FileNameVector);

if (comm.Rank != 0)

comm.Send(0, comm.Rank - 1, MatrixHeight \* MatrixLenght);

for (int i = 0, j = 0, a = 0, s = 0; a < MatrixText.Length; a++)

{

if (MatrixText[a] == ' ')

{

if (MatrixText[a - 1] != ' ')

{

Matrix[i, j] = Convert.ToInt32(MatrixText.Substring(s, a - s));

j++;

}

s = a + 1;

}

if (MatrixText[a] == '\n')

{

if (MatrixText[a - 1] != '\n')

{

Matrix[i, j] = Convert.ToInt32(MatrixText.Substring(s, a - s));

i++;

j = 0;

}

s = a + 1;

}

}

for (int i = 0, a = 0, s = 0; a < VectorText.Length; a++)

{

if (VectorText[a] == '\n')

{

Vector[i] = Convert.ToInt32(VectorText.Substring(s, a - s));

i++;

s = a + 1;

}

}

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

{

CheckResult[i] = 0;

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

{

CheckResult[i] += Matrix[i, j] \* Vector[j];

}

}

int[] R = new int[EndX[comm.Rank] - StartX[comm.Rank] + 1];

for (int i = StartX[comm.Rank]; i <= EndX[comm.Rank]; i++)

{

R[i - StartX[comm.Rank]] = 0;

for (int j = StartY[comm.Rank]; j <= EndY[comm.Rank]; j++)

{

R[i - StartX[comm.Rank]] += Matrix[i, j] \* Vector[j];

}

}

if (comm.Rank != 0)

{

for (int i = 0; i < R.Length; i++)

{

comm.Send(R[i], 0, i);

}

}

else

{

for (int j = StartX[0]; j <= EndX[0]; j++)

{

Result[j] = R[j - StartX[0]];

}

mulVectorToMatrix(Vector,Matrix);

Console.WriteLine("The result is correct");

}

}

Console.ReadKey();

}

static void RandomGenericValue()

{

Random rand = new Random();

string Vector = "";

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

{

Vector += rand.Next(0, 9) + "" + (char)13 + "" + (char)10;

}

File.WriteAllText(FileNameVector, Vector);

string Matrix = "";

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

{

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

{

Matrix += 1;

if (j != MatrixLenght - 1)

Matrix += " ";

}

Matrix += (char)13 + "" + (char)10;

}

File.WriteAllText(FileNameMatrix, Matrix);

}

static void mulVectorToMatrix(int[] vec, int[,] mat)

{

string Result = "";;

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

{

int res = 0;

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

{

//матриця на стовбець

res += mat[i, j] \* vec[j];

}

Result += res.ToString() + "\n";

}

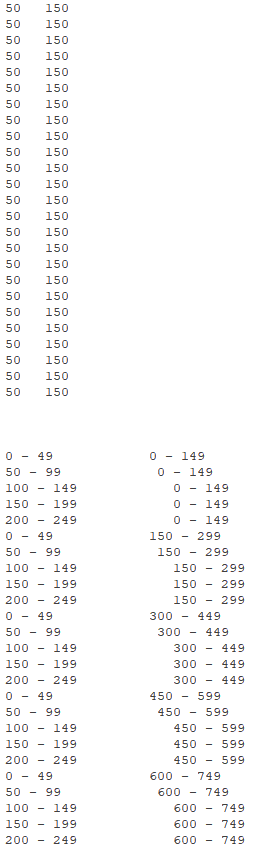
File.WriteAllText(FileNameResult, Result);

}

}

}

**Результат роботи програми**



*Рис.2 Розбиття матриці*

На рис.2 показано розрахунок розбиття матриці на блоки. Спочатку йде розбиття розмірів для кожного процесору, а далі – початкові та кінцеві координати в матриці.

Вхідні дані матриці, вектора і результату знаходяться в файлах Matrix.txt, Vector.txt і Result.txt відповідно, які знаходяться в папкі проекту.

**Висновок:**

На даній лабораторній роботі розроблено алгоритм паралельного перемноження матриці на вектор при блоковому розбитті вхідних даних. Виконано його програмну реалізацію з використанням МРІ. Розроблено схему інформаційної взаємодії між підзадачами та виконано їх масштабування на задану кількість проесорів системи. Обчислено кількість елементів та операцій для кожного процесора.