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: студент .гр. КІ-33

Харченко М.В.

Прийняв: асистент каф. ЕОМ

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** |
| 10 | 240 | 120 | стрічкове(гор) | 15 |

**Хід роботи:**

**Підготовка**

**Розбиття матриці**

При горизонтальному способі розбиття даних (розбиття матриці на горизонтальні смуги) вхідна матриця буде мати такий вигляд:



*Рис.1. Розбиття вхідної матриці на блоки для 10 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 = 240 x 120 / 15 + 240 + 120 = 77 елементів;

Кількість операцій визначається на основі формули:

*T1 = m·(2n-1) = 240 \* 239 = 57360*

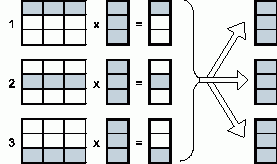
та становить 5 590операцій для кожного процесора.

**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. 2.



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

**Розробка програми з використанням МРІ**

Програма реалізує логіку роботи алгоритму, послідовно викликає необхідні підпрограми, написана на мові програмування c++ з використанням фреймворка qt із використання бібліотеки omp.h

#include <iostream>

#include <stdlib.h>

#include "mpi.h"

#include <Windows.h>

#include<ctime>

#include <fstream>

#define debug\_tt 1

using namespace std;

#define namber 2000

const int route = 0;

const int circle[] = { 0, 1, 2, 3, 4, 5, 6, 7, 8,9,10,11,12,13,14 };

const int procNumber = 15;

const int N1 = 240;

const int N2 = 120;

const int N3 = 1;

int procRank;

const int root = 0;

MPI\_Status Status;

double start\_time; // начальное время

double end\_time; // конечное время

double start2\_time; // начальное время

double end2\_time; // конечное время

double rez\_time; // начальное время

double rez2\_time; // конечное время

unsigned long int search\_time;

struct A {

int row;

int n1;

int n2;

double arr[N1 / procNumber + 1][N2];

};

struct B {

int col;

int n2;

int n3;

double arr[N2][N3 / procNumber + 1];

};

struct R {

int n1;

int n3;

double arr[N1 / procNumber + 1][N3];

};

double matrixA[N1][N2] = { 0 };

double matrixB[N2][N3] = { 0 };

double Res[N1][N3] = { 0 };

A subA[procNumber];

B subB[procNumber];

A procA;

B procB;

B procBtmp;

R procRes;

R procResTmp;

R result[procNumber];

int NextProc;

int PrevProc;

int procSize;

//Збір

void DataReplicetion() {

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

if (procRank == circle[i]) {

if (procRank != root) {

MPI\_Send(&procRes, sizeof(R), MPI\_BYTE, 0, 0, MPI\_COMM\_WORLD);

}

}

}

if (procRank == root) {

result[root] = procRes;

for (int k = 0; k < procNumber - 1; k++) {

MPI\_Recv(&result[circle[1 + k]], sizeof(procRes), MPI\_BYTE, circle[k + 1], 0, MPI\_COMM\_WORLD, &Status);

}

}

}

//початкова розсилка

void DataDistribution() {

if (procRank == root) {

start2\_time = clock();

procA = subA[root];

procB = subB[root];

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

if (i != root) {

MPI\_Send(&subA[circle[i]], sizeof(subA[circle[i]]), MPI\_BYTE, circle[i], 0, MPI\_COMM\_WORLD);

MPI\_Send(&subB[0], sizeof(subB[circle[i]]), MPI\_BYTE, circle[i], 0, MPI\_COMM\_WORLD);

}

}

MPI\_Status Status;

if (procRank != root) {

MPI\_Recv(&procA, sizeof(procA), MPI\_BYTE, root, 0, MPI\_COMM\_WORLD, &Status);

MPI\_Recv(&procB, sizeof(procB), MPI\_BYTE, root, 0, MPI\_COMM\_WORLD, &Status);

}

}

// перемноження

void DataSend() {

procRes.n1 = procA.n1;

procRes.n3 = N3;

for (int i = 0; i < procA.n1; i++)

for (int j = 0; j < procB.n3; j++)

for (int k = 0; k < procA.n2; k++)

procRes.arr[i][j + procB.col] += procA.arr[i][k] \* procB.arr[k][j];

}

int main(int argc, char\* argv[])

{

MPI\_Init(&argc, &argv);

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &procRank);

MPI\_Comm\_size(MPI\_COMM\_WORLD, &procSize);

if (procSize != procNumber) {

cout << "Must be 15 process" << endl;

MPI\_Finalize();

return 1;

}

int format = 2;

if (procRank == root) {

int check = 0;

double numb;

while (true) {

cout << "Random 1-9 - enter 1\nEnter number - enter 2\n";

cin >> check;

if (check == 1 | check == 2)

break;

}

if (check == 2) {

cout << "Enter number\n";

cin >> numb;

}

//заповнення А

ofstream fo;

for (int i = 0; i < N1; i++)

for (int j = 0; j < N2; j++) {

if (check == 1)

matrixA[i][j] = rand() % 9 + 1;

else

matrixA[i][j] = numb;

}

cout << "Matrix A" << endl;

if (!debug\_tt) {

for (int i = 0; i < N1; i++) {

for (int j = 0; j < N2; j++)

cout << matrixA[i][j] << " ";

cout << endl;

}

fo.open("MatrixA.txt", std::ofstream::out);

for (int i = 0; i < N1; i++)

{

fo << i + 1 << ": ";

for (int j = 0; j < N2; j++) {

fo.width(2);

fo << matrixA[i][j] << " ";

}

fo << "\r\n";

}

fo.close();

//заповнення В

for (int i = 0; i < N2; i++)

for (int j = 0; j < N3; j++)

{

if (check == 1)

matrixB[i][j] = rand() % 9 + 1;

else

matrixB[i][j] = numb;

}

cout << endl << "Matrix B" << endl;

for (int i = 0; i < N2; i++) {

for (int j = 0; j < N3; j++)

cout << matrixB[i][j] << " ";

cout << endl;

}

fo.open("MatrixB.txt", std::ofstream::out);

for (int i = 0; i < N2; i++)

{

fo << i + 1 << ": ";

for (int j = 0; j < N3; j++) {

fo.width(2);

fo << matrixB[i][j] << " ";

}

fo << "\r\n";

}

fo.close();

}

// перемноження матриць

start\_time = clock();

for (int i = 0; i < N1; i++)

for (int j = 0; j < N3; j++)

for (int k = 0; k < N2; k++)

Res[i][j] += matrixA[i][k] \* matrixB[k][j];

end\_time = clock();

rez\_time = end\_time - start\_time;

int max = Res[0][0];

if (!debug\_tt) {

cout << "Rez serial: " << endl;

for (int i = 0; i < N1; i++) {

for (int j = 0; j < N3; j++) {

cout << Res[i][j] << " ";

if (Res[i][j] > max)

max = Res[i][j];

}

cout << endl;

}

while (max > (pow(10, format))) {

format++;

}

format++;

fo.open("MatrixC\_serial.txt", std::ofstream::out);

for (int i = 0; i < N1; i++)

{

fo << i + 1 << ": ";

for (int j = 0; j < N3; j++) {

fo.width(format);

fo << Res[i][j] << " ";

}

fo << "\r\n";

}

fo.close();

}

//розбиття на підматриці

//А

int rowsSum = 0;

int restRows = N1;

int restProc = procNumber;

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

{

subA[i].row = rowsSum;

subA[i].n1 = restRows / restProc;

subA[i].n2 = N2;

for (int k = 0; k < subA[i].n1; k++)

for (int j = 0; j < subA[i].n2; j++)

subA[i].arr[k][j] = matrixA[k + rowsSum][j];

rowsSum += subA[i].n1;

restRows -= restRows / restProc;

restProc--;

}

//В

rowsSum = 0;

restProc = procNumber;

restRows = N3;

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

{

subB[i].col = rowsSum;

subB[i].n2 = N2;

subB[i].n3 =1 ;// restRows / restProc;

for (int k = 0; k < subB[i].n2; k++)

for (int j = 0; j < subB[i].n3; j++)

subB[i].arr[k][j] = matrixB[k][j + rowsSum];

rowsSum += subB[i].n3;

restRows -= restRows / restProc;

restProc--;

}

}

DataDistribution();

DataSend();

DataReplicetion();

if (procRank == root) {

end2\_time = clock();

rez2\_time = end2\_time - start2\_time;

cout << endl << "Paralel rez" << endl;

if (!debug\_tt) {

for (int k = 0; k < procNumber; k++)

for (int i = 0; i < result[k].n1; i++) {

for (int j = 0; j < result[k].n3; j++) {

cout << result[k].arr[i][j] << " ";

}

cout << endl;

}

ofstream fo;

fo.open("MatrixC\_Paralel.txt", std::ofstream::out);

for (int i = 0; i < N1; i++)

{

fo << i + 1 << ": ";

for (int j = 0; j < N3; j++) {

fo.width(format);

fo << Res[i][j] << " ";

}

fo << "\r\n";

}

fo.close();

}

cout << endl << "Result time serial: " << rez\_time << endl;

cout << endl << "Result time paralel: " << rez2\_time << endl;

}

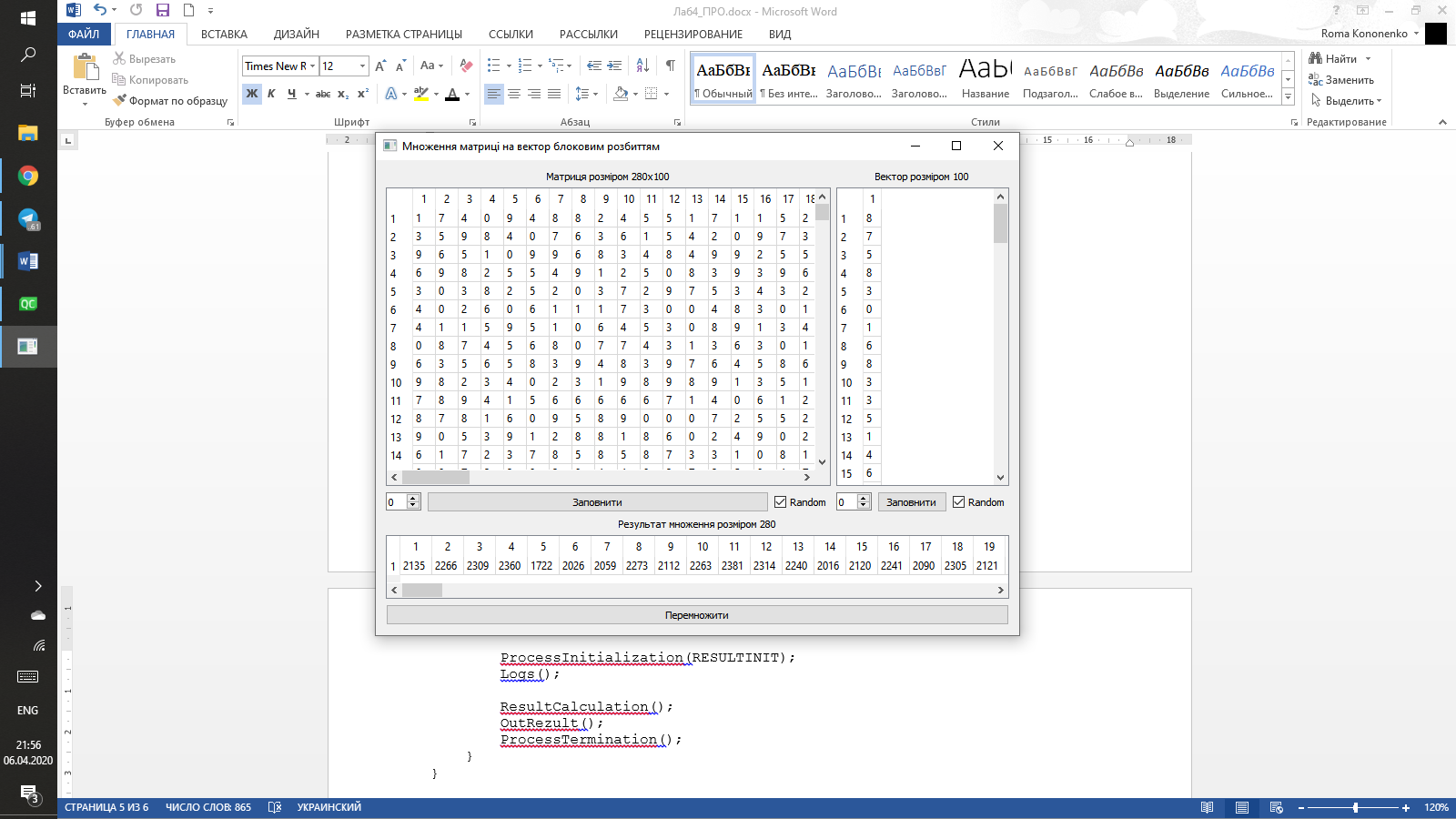
MPI\_Finalize();

exit(0);

return 0;

}

**Результат виконання програми:**



**Висновок:**

На даній лабораторній роботі розроблено алгоритм паралельного перемноження матриці на вектор при стрічковому горизонтальному 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.