**Міністерство освіти і науки України**

**Національний університет «Львівська політехніка»**



Звіт

до лабораторної роботи № 4

з дисципліни: «Паралельно та розподілені обчислення»

на тему:« Паралельны алгоритми множення матриці на вектор»

Виконав:

ст.гр. КІ-34

Маковський В. І.

Перевірив:

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** |
| 8 | 100 | 280 | стрічкове(гор) | 10 |

**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 = 100 x 280 / 28 + 280 + 100 = 1380 елементів;

Кількість операцій визначається на основі формули:

*T1 = m·(2n-1) = 100 \* 559 = 55900*

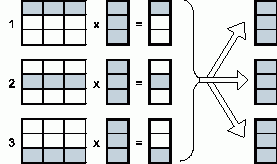
та становить 5 590операцій для кожного процесора.

**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 "mainwindow.h"

#include "ui\_mainwindow.h"

#include "QDebug"

const int MATRIXINIT = 1;

const int VECTORINIT = 2;

const int RESULTINIT = 3;

const int ROWS = 280;

const int COLS = 100;

const int PROCNUM = 10;

QVector<int> pMatrix;

QVector<int> pVector;

QVector<int> pResult;

MainWindow::**MainWindow**(QWidget \*parent) : QMainWindow(parent), ui(new Ui::MainWindow)

{

ui->setupUi(this);

}

MainWindow::~***MainWindow***()

{

delete ui;

}

inline void MainWindow::**DataInitialization**(int rows, int cols)

{

bool matrixInit = (rows > 1) ? true : false;

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

{

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

{

QTableWidgetItem \*item = new QTableWidgetItem();

if(matrixInit)

{

pMatrix[i\*cols + j] = (ui->RandMatA->isChecked())

? rand() % 10

: ui->NumMatA->value();

item->setText(QString::number(pMatrix[i\*cols + j]));

ui->MatrixA->setItem(i, j, item);

}

else

{

pVector[i\*cols + j] = (ui->RandVecB->isChecked())

? rand() % 10

: ui->NumVecB->value();

item->setText(QString::number(pVector[i\*cols + j]));

ui->VectorB->setItem(i, j, item);

}

}

}

}

void MainWindow::**Logs**()

{

int rows = ROWS / PROCNUM;

int procNum = PROCNUM;

QFile file("file.txt");

if (!file.*open*(QIODevice::WriteOnly))

{

qDebug() << "Ошибка при открытии файла";

}

QTextStream out(&file);

QString matrix\_s = "";

QString vector\_s = "";

QString result\_s = "";

QVector<int> result;

result.resize(ROWS);

// "send" vector

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

{

vector\_s += QString::number(pVector[i]) + "\r\n";

}

for(int I = 0, res = 0; I < ROWS; I += rows)

{

out << "\r\nProcNum: " + QString::number(--procNum) + "\r\n";

for(int i = I; i < I + rows; ++i, ++res)

{

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

{

matrix\_s += QString::number(pMatrix[i\*COLS + j]) + "\t";

result[res] += pMatrix[i\*COLS + j]\*pVector[j];

}

matrix\_s += "\r\n";

}

out<<"Line Matrix:\r\n" + matrix\_s;

out<<"Vector:\r\n" + vector\_s;

matrix\_s = "";

}

for(int i = 0; i < ROWS; i += rows)

{

out << "\r\nProcNum: " + QString::number(procNum++) + "\r\n";

for(int j = i; j < i +rows; ++j)

{

result\_s += QString::number(result[j]) + "\t";

}

out << "Part Result:\r\n" + result\_s + "\r\n";

result\_s = "";

}

file.*close*();

}

void MainWindow::**ProcessInitialization**(int WhatInitialize)

{

if(WhatInitialize == MATRIXINIT)

{

ui->MatrixA->setRowCount(ROWS);

ui->MatrixA->setColumnCount(COLS);

ui->MatrixA->setItemDelegate(new tabledelegate);

pMatrix.resize(COLS\*ROWS);

}

if(WhatInitialize == VECTORINIT)

{

ui->VectorB->setRowCount(COLS);

ui->VectorB->setColumnCount(1);

ui->VectorB->setItemDelegate(new tabledelegate);

pVector.resize(COLS);

}

if(WhatInitialize == RESULTINIT)

{

ui->Result->setRowCount(1);

ui->Result->setColumnCount(ROWS);

ui->Result->setItemDelegate(new tabledelegate);

pResult.resize(ROWS);

pResult.fill(0);

}

}

void **ProcessTermination**()

{

pResult.clear();

}

void **ResultCalculation**()

{

int NestedThreadsNum = 10;

omp\_set\_num\_threads(NestedThreadsNum);

omp\_set\_nested(true);

#pragma omp parallel for

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

{

double ThreadResult = 0;

#pragma omp parallel for reduction(+:ThreadResult)

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

{

ThreadResult += pMatrix[i\*COLS + j] \* pVector[j];

}

pResult[i] = ThreadResult;

}

}

void MainWindow::**OutRezult**()

{

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

{

QTableWidgetItem \*item = new QTableWidgetItem();

item->setText(QString::number(pResult[i]));

ui->Result->setItem(0, i, item);

}

}

void MainWindow::**on\_Multiply\_clicked**()

{

if(!pMatrix.isEmpty() && !pVector.isEmpty())

{

ProcessInitialization(RESULTINIT);

Logs();

ResultCalculation();

OutRezult();

ProcessTermination();

}

}

void MainWindow::**on\_Fill\_MatrixA\_clicked**()

{

pMatrix.clear();

ProcessInitialization(MATRIXINIT);

DataInitialization(ROWS, COLS);

}

void MainWindow::**on\_Fill\_VectorB\_clicked**()

{

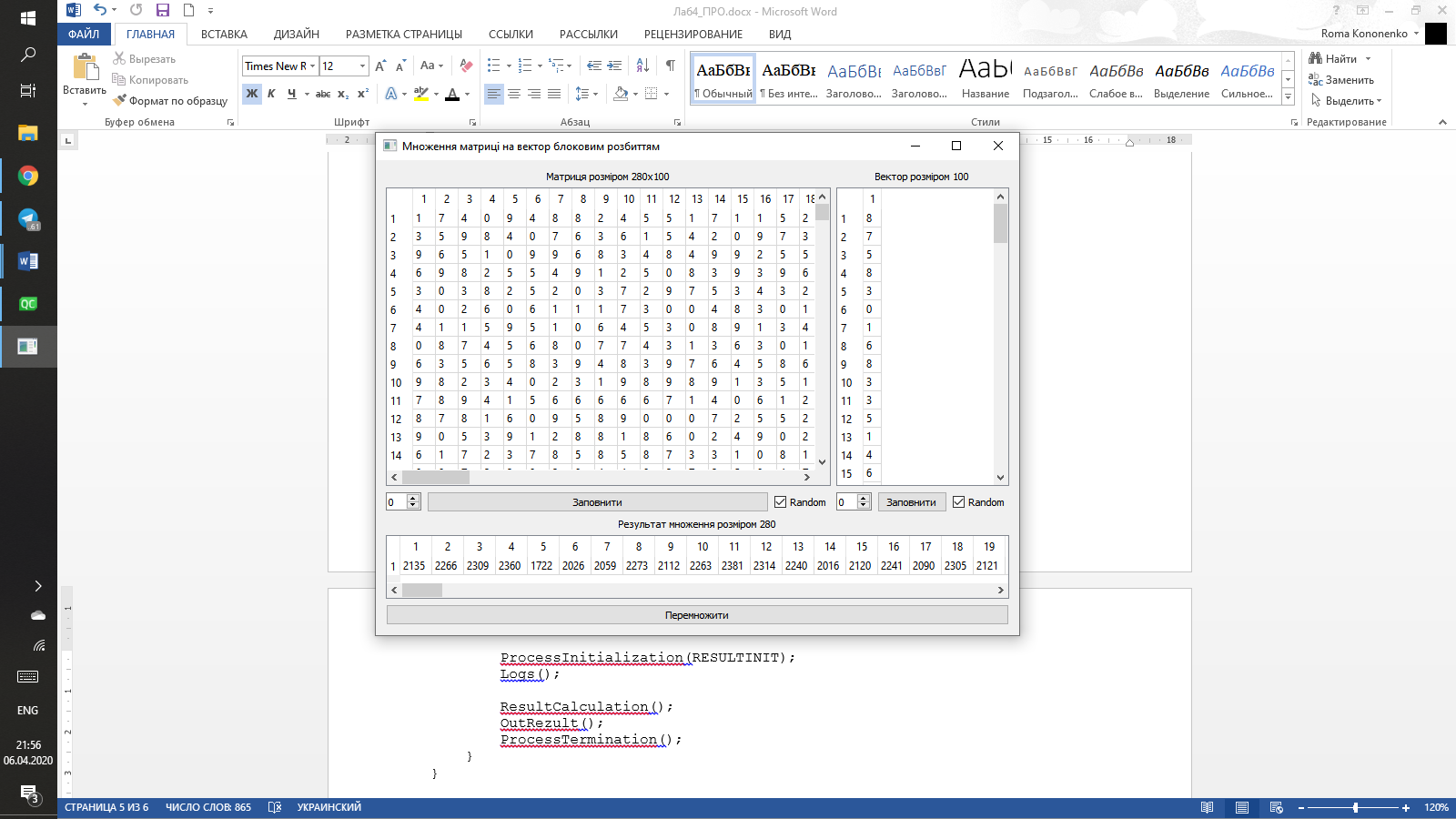
pVector.clear();

ProcessInitialization(VECTORINIT);

DataInitialization(1, COLS);

}

**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.