**Лабораторна робота 5. Приклад розробки складної програми за допомогою MPI**

**Мета роботи:**

• Продемонструвати процес створення програми для вирішення реальної математичної задачі на кластерній системі.

• Показати комплексне використання раніше вивчених функцій MPI.

**5.1** **Математичний апарат**

**Постановка задачі**

Двовимірне рівняння Пуассона являє собою диференційне рівняння у часткових похідних виду:



Очевидно, що розв'язати його у аналітичному вигляді немає можливості. Одним із можливих методів розв'язання є застосування методу кінцевих різниць.

**5.1.1** **Метод кінцевих різниць**

Метод кінцевих різниць передбачає дискретизацію диференціальних рівнянь на так званих прямокутних координатних сітках, тобто на сітках, елементарні комірки яких представляють собою прямокутники для двовимірних або паралелепіпеди для трьохвимірних задач.

**Кінцево-різничні сітки**

Розглянемо одновимірну область , що являє собою відрізок . Розіб'ємо цей відрізок точками  на  рівних частин довжиною  кожна. Множина точок  називається рівномірною одномірною координатною сіткою, а число  – кроком сітки.

Відрізок  можна розбити на  частин, вводячи довільні точки .

Координатна сітка  буде мати крок , що залежить від номера  вузла . Якщо  хоча б для одного номера , координатна сітка  називається нерівномірною (рис. 5.1).



Figure 5.1: Одновимірна сітка

Аналогічно двовимірною сіткою називають множину точок . Нерівномірна двовимірна сітка являє собою множину  (рис. 5.2).

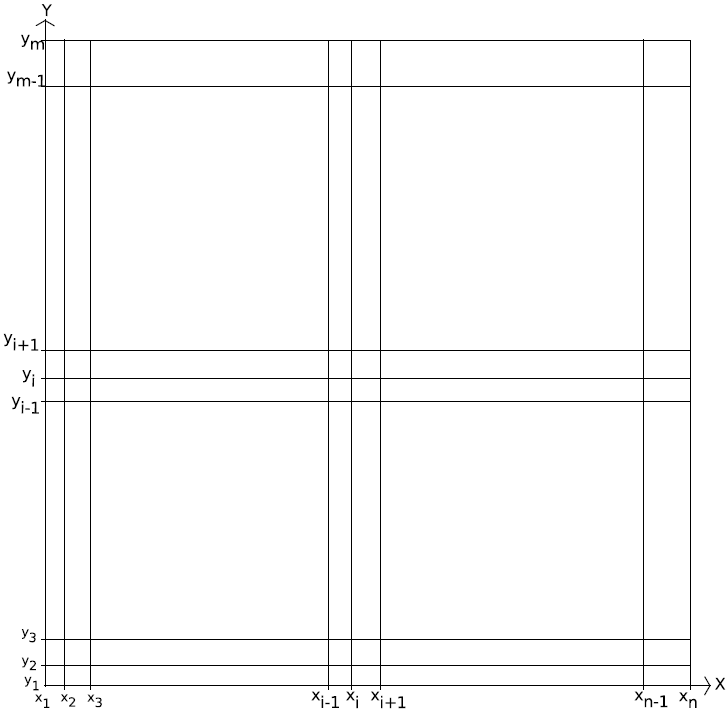


Figure 5.2: Двовимірна сітка

**Сіткові функції, кінцеві різниці**

Введення для області  координатної сітки  передбачає що значення всіх змінних та їх похідних розглядаються тільки у вузлах цієї сітки. З ціллю виконання цієї умови всі змінні задачі сітковими функціями, а похідні будь-якого порядку --- кінцевими різницями.

Нехай для деякої області  задана сітка . Тоді функцію  дискретного аргументу  називають сітковою функцією, визначеною на сітці .

Будь-якій неперервній функції , заданій в області , можна поставити у відповідність сіткову функцію  (для зручності будемо позначати ), задану на сітці  (спроектувати функцію  на сітку ), приймаючи до уваги певне правило співвідношення. Наприклад:





де  – координати серединних точок на відповідних кроках координатної сітки, що визначаються виразами:









Слід мати на увазі, що одна й та ж сіткова функція, задана на двох різних сітках, що маються спільні вузли, не обов'язково буде мати в цих вузлах однакові значення.

За визначенням похідна функції неперервного аргументу  у точці  є ліміт відношення приросту функції до приросту аргументу, коли приріст аргументу близиться до нуля:



Знехтувавши границею, похідну функції неперервного аргументу можна приблизно замінити (апроксимувати) різницевим виразом, заданим на відповідній сітковій функції . Дана апроксимація може бути виконана декількома способами:









де  – кінцеві різниці координат, що визначаються виразами:





Для кожного з перетворень характерна похибки апроксимації, що прямує до нуля коли крок сітки прямує до нуля.

Похідні другого порядку апроксимуються наступним чином:





У випадку коли сітка рівномірна вирази спрощуються:





**5.2** **Аналіз задачі**

Застосувавши вище приведений метод до нашої задачі, ми можемо визначити структуру обчислювальної моделі та формули, за якими будемо вести розрахунок.

Структура обчислювальної моделі зображена на рисунку 5.3. Сітка дискретизації квадратна та рівномірна.

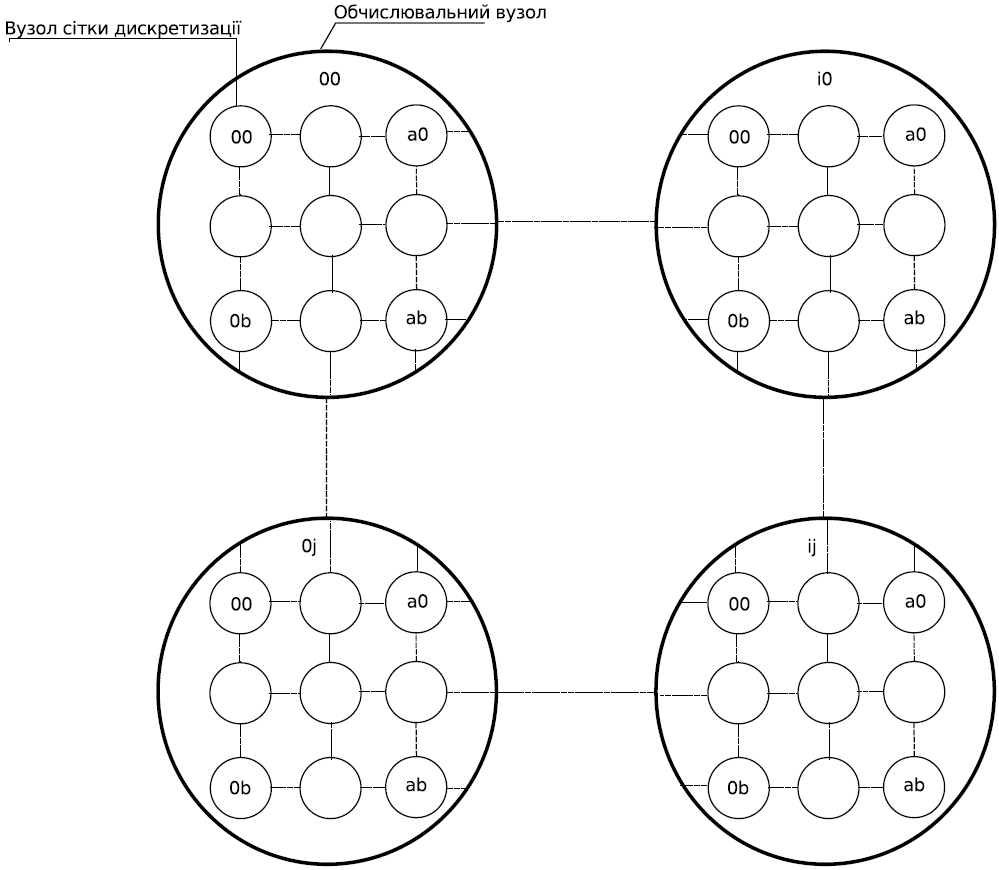


Figure 5.3: Структура обчислювальної моделі

У кожному обчислювальному вузлі буде послідовно розраховуватися матриця точок сітки дискретизації. Реальні вузли будуть працювати паралельно між собою та обмінюватися даними.

Нехай початкові умови функції всередині області , а на границі області .

Тоді головна ітеративна формула матиме вигляд:



де:

•  – координати вузла сітки дискретизації;

•  – номер ітерації;

•  – крок у просторі, що визначається за формулою ;

•  – кількість вузлів сітки дискретизації вздовж однієї з координат;

•  – значення правої частини диференційного рівняння у вузлі сітки дискретизації;

•  – різницевий оператор, що визначається за формулою:



Локальним критерієм завершення алгоритму слугує досягнення заданої точності:



де  задається до початку розрахунку.

**5.3** **Аналіз програмної реалізації**

Спрощено алгоритм основного циклу програми виглядає наступним чином:

1. Зробити розрахунки для тих вузлів сітки дискретизації, що не потребують даних від сусідніх процесів.

2. Отримати колонку сітки дискретизації від лівого сусіднього процесу, якщо він є.

3. Передати крайню праву колонку сітки дискретизації до правого сусіднього процесу, якщо він є.

4. Отримати колонку сітки дискретизації від правого сусіднього процесу, якщо він є.

5. Передати крайню ліву колонку сітки дискретизації до лівого сусіднього процесу, якщо він є.

6. Отримати рядок сітки дискретизації від верхнього сусіднього процесу, якщо він є.

7. Передати нижній рядок сітки дискретизації до нижнього сусіднього процесу, якщо він є.

8. Отримати рядок сітки дискретизації від нижнього сусіднього процесу, якщо він є.

9. Передати верхній рядок сітки дискретизації до верхнього сусіднього процесу, якщо він є.

10. Завершити розрахунки для інших вузлів, які не були розраховані у пункті 1.

11. Перевірити локальні критерії завершення алгоритмів. Якщо усі вузли сітки у процесі завершили свою – завершити розрахунки у процесі.

12. Якщо всі процеси завершили розрахунки – завершити алгоритм.

Розглянемо деякі проблеми, зв'язані з реалізацією цього алгоритму:

• Для передачі стовпців матриці застосуємо вже відомий механізм наслідування типів. У цьому випадку нам підійде конструктор MPI\_Type\_vector.

MPI\_Type\_vector(count, blocklength, stride, oldtype, newtype)

IN count кількість блоків

IN blocklength кількість елементів у кожному блоці

IN stride відстань між початками кожного блоку,

у кількості елементів

N oldtype тип, від якого наслідуємо

OUT newtype новий тип

Отже, щоб виділити стовпець з матриці ми повинні записати наступний конструктор:

MPI\_Type\_vector(<висота матриці>,

1,

<ширина матриці>,

MPI\_DOUBLE,

&new\_type);

Таким чином, викликаючи

MPI\_Send(<Покажчик на перший елемент стовпця>,

1,

new\_type,

<ціль передачі>,

<ідентифікатор передачі>,

MPI\_COMM\_WORLD);

ми передамо увесь стовпець.

• Аналогічна проблема виникає при початковому розподілі координат вузлів сітки дискретизації по процесам. Фактично нам потрібно відправити квадратну підматрицю. Реалізація змінюється тільки у ширині блоку і їх кількості при конструюванні типу.

MPI\_Type\_vector(<висота блоку, що відправляємо>,

<ширина блоку, що відправляємо>,<ширина матриці>,

MPI\_DOUBLE,&new\_type);

• Попри те, що процес закінчив розрахунок, він усе одно повинен взаємодіяти з сусідніми процесами, щоб ті теж могли завершити розрахунки. Для цього треба реалізувати подвійну перевірку – умови зупинки розрахунку процесу та умови глобального кінця розрахунку. У програмі спочатку виконується перевірка усіх локальних умов закінчення алгоритму. Якщо усі вузли сітки дискретизації, що належать до процесу, закінчили роботу, то встановлюється ознака закінчення обчислень у процесі. Після цього усі ознаки закінчення обчислень у процесах збираються нульовим процесом и над ними виконується операція логічного <<І>> за допомогою MPI\_Reduce.

MPI\_Reduce(&node\_stop,

&global\_stop,

1,

MPI\_SHORT,

MPI\_LAND,

0,

MPI\_COMM\_WORLD);

Результат цієї операції є глобальною ознакою закінчення алгоритму, оскільки він матиме значення true тільки коли усі процеси закінчили обчислення. Після цього глобальна ознака закінчення обчислення розповсюджується між усіма процесами за допомогою MPI\_Bcast.

**6.3.1** **Код програми**

|  |
| --- |
| Увага! Перед запуском прикладів коду на кластері ознайомтесь з інструкціями в додатку B. |

#define \_GNU\_SOURCE

#include <mpi.h>

#include <stdio.h>

#include <stdarg.h>

#include <stdlib.h>

#include <stdbool.h>

#include <math.h>

/\* Опис типів,структур та функцій \*/

struct my\_matrix /\* Структура, що описує матриці \*/

{

int rows; /\* Кількість строк у матриці \*/

int cols; /\* Кількість стовпців у матриці \*/

double \*data; /\* Ссилка на матрицю, розвернуту у одномірний масив\*/

};

typedef struct my\_matrix my\_matrix;

MPI\_Datatype node\_mat\_t; /\* Тип, що забеспечує розповсюдження частин сітки

дискретизації по матриці процессів\*/

MPI\_Datatype mat\_col; /\* Тип, що забеспечує передачу стовпця матриці \*/

double func(double X,double Y);/\* Повертає значення правої частини

диференційного рівняння у точці X,Y\*/

void fatal\_error(const char \*message, int errorcode);/\* Функція виведення

помилок вводу-виводу \*/

struct my\_matrix \*matrix\_alloc(int rows, int cols, double initial);/\* Фунція,

що забеспечує виділення пам'яті та ініціалізацію матриці\*/

void matrix\_print(const char \*filename, struct my\_matrix \*mat);/\* Фунція, що

забеспечує виведення матриці у файл\*/

struct my\_matrix \*read\_matrix(const char \*filename);/\* Функція, що забеспечує

зчитування матриці із файлу\*/

/\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*/

/\* Опис змінних \*/

int np; /\* Кількість процесів \*/

int rank; /\* Ранг процесу у MPI\_COMM\_WORLD \*/

int nodes\_width; /\* Ширина матриці процессів \*/

int node\_X; /\* Координата Х у матриці процессів \*/

int node\_Y; /\* Координата У у матриці процессів \*/

int total\_width; /\* Ширина матриці вузлів сітки дискретизації\*/

int points\_per\_node; /\* Ширина частини матриці вузлів сітки дискретизації,

що припадає на кожен процес\*/

const char\* inpfileX="inpX.csv"; /\* Ім'я файлу з якого вводяться горизонтальні координати сітки дискретизації \*/

const char\* inpfileY="inpY.csv"; /\* Ім'я файлу з якого вводяться вертикальні

координати сітки дискретизації \*/

const char\* outpfile="outp.csv";/\* Ім'я файлу куди виводиться результат \*/

my\_matrix\* allcoords\_X;/\* Матриця реальних горизонтальних координат вузлів

сітки дискретизації \*/

my\_matrix\* allcoords\_Y;/\* Матриця реальних вертикальних координат вузлів сітки дискретизації \*/

my\_matrix\* total\_mat;/\* Повна сітка дисретизації, що виводиться у процессі 0\*/

my\_matrix\* coords\_X;/\* Частина матриці реальних координат сітки дискретизації, яка зберігається у кожному з процессів\*/

my\_matrix\* coords\_Y;/\* Частина матриці реальних координат сітки дискретизації, яка зберігається у кожному з процессів\*/

my\_matrix\* node\_mat;/\* Частина сітки дискретизації, яка зберігається у кожному з процессів\*/

my\_matrix\* f\_xy;/\* Масив значень функції в узлах сітки дискретизації\*/

MPI\_Status stat;/\* Змінна, у яку повертається статус прийому повідомленнь \*/

double epsilon=0.001;/\* Точність рішення\*/

double omega=0.1;/\*\*/

double h;/\* Крок сітки дискретизації \*/

bool\* local\_stop;/\* Масив для запам'ятовування локальних умов зупинки \*/

bool node\_stop;/\* Ознака завершення обчислень у процесі\*/

bool global\_stop;/\* Глобальна ознака завершення обчислень\*/

double\* left\_col;/\* Стовпці, що процесс буде приймати з лівого та правого

процессів під час ітерацій \*/

double\* right\_col;

double\* top\_row;/\* Рядки, що процесс буде приймати з верхнього та нижнього

процессів під час ітерацій \*/

double\* bot\_row;

/\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*/

/\* Головна программа \*/

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

{

MPI\_Init(&argc,&argv); /\* Ініціалізуємо середовище \*/

MPI\_Comm\_size(MPI\_COMM\_WORLD,&np); /\* Отримуємо розмір MPI\_COMM\_WORLD \*/

MPI\_Comm\_rank(MPI\_COMM\_WORLD,&rank); /\* Отримуємо ранг процеса у MPI\_COMM\_WORLD\*/

nodes\_width=sqrt(np); /\* Отримуємо ширину матриці процесів\*/

node\_X=rank%nodes\_width; /\* Отримуємо горизонтальну кординату процесу у матриці \*/

node\_Y=rank/nodes\_width; /\* Отримуємо вертикальну кординату процесу у матриці \*/

if (rank==0){

allcoords\_X=read\_matrix(inpfileX);/\* Вводимо координати \*/

allcoords\_Y=read\_matrix(inpfileY);/\* Вводимо координати \*/

total\_width=allcoords\_X->rows; /\* Отримуємо ширину сітки дискретизації \*/

}

MPI\_Bcast(&total\_width,1,MPI\_INT,0,MPI\_COMM\_WORLD); /\* Розповсюджуємо ширину сітки дискретизації \*/

h=1.0/total\_width;/\* Визначаємо крок сітки дискретизації \*/

points\_per\_node=total\_width/nodes\_width;

/\* Створюємо тип, що відповідає частині матриці, яка буде передаватися \*/

MPI\_Type\_vector(points\_per\_node,points\_per\_node,total\_width,MPI\_DOUBLE,&node\_mat\_t);

MPI\_Type\_commit(&node\_mat\_t); /\* Регіструємо тип \*/

coords\_X=matrix\_alloc(points\_per\_node,points\_per\_node,0.0);

coords\_Y=matrix\_alloc(points\_per\_node,points\_per\_node,0.0);

if(rank==0){/\* Процесс 0 розсилає реальні координати сітки дискретизації\*/

for(int i=0;i<nodes\_width;i++){

for(int j=0;j<nodes\_width;j++){

if(!(i==0&&j==0)){

MPI\_Send(allcoords\_X->data+i\*total\_width\*points\_per\_node+j\*points\_per\_node

,1,node\_mat\_t,i\*nodes\_width+j,0,MPI\_COMM\_WORLD);

MPI\_Send(allcoords\_Y->data+i\*total\_width\*points\_per\_node+j\*points\_per\_node

,1,node\_mat\_t,i\*nodes\_width+j,0,MPI\_COMM\_WORLD);

}

}

}

for(int i=0;i<points\_per\_node;i++){

for(int j=0;j<points\_per\_node;j++){

coords\_X->data[i\*points\_per\_node+j]=allcoords\_X->data[i\*total\_width+j];

coords\_Y->data[i\*points\_per\_node+j]=allcoords\_Y->data[i\*total\_width+j];

}

}

}

else{/\* Інші процесси їх приймають \*/

my\_matrix\* temp\_X=matrix\_alloc(total\_width,points\_per\_node,0);

my\_matrix\* temp\_Y=matrix\_alloc(total\_width,points\_per\_node,0);

MPI\_Recv(temp\_X->data,1,node\_mat\_t,0,0,MPI\_COMM\_WORLD,&stat);

MPI\_Recv(temp\_Y->data,1,node\_mat\_t,0,0,MPI\_COMM\_WORLD,&stat);

for(int i=0;i<points\_per\_node;i++){

for(int j=0;j<points\_per\_node;j++){

coords\_X->data[i\*points\_per\_node+j]=temp\_X->data[i\*total\_width+j];

coords\_Y->data[i\*points\_per\_node+j]=temp\_Y->data[i\*total\_width+j];

}

}

}

/\* Ініціалізуємо і встановлюємо початкові значення внутрішнього діапазону \*/

node\_mat=matrix\_alloc(points\_per\_node,points\_per\_node,0);

/\* Встановлюємо початкові значення граничних вузлів сітки дискретизації \*/

if(node\_X==0){

for(int i=0;i<points\_per\_node\*points\_per\_node;i+=points\_per\_node){

node\_mat->data[i]=(1+3\*M\_PI)\*sin(3\*M\_PI\*(coords\_X->data[i]+coords\_Y->data[i]));

}

}

if(node\_X==nodes\_width-1){

for(int i=points\_per\_node-1;i<points\_per\_node\*points\_per\_node;i+=points\_per\_node){

node\_mat->data[i]=(1+3\*M\_PI)\*sin(3\*M\_PI\*(coords\_X->data[i]+coords\_Y->data[i]));

}

}

if(node\_Y==0){

for(int i=0;i<points\_per\_node;i++){

node\_mat->data[i]=(1+3\*M\_PI)\*sin(3\*M\_PI\*(coords\_X->data[i]+coords\_Y->data[i]));

}

}

if(node\_Y==nodes\_width-1){

for(int i=(points\_per\_node-1)\*points\_per\_node;i<points\_per\_node\*points\_per\_node;i++){

node\_mat->data[i]=(1+3\*M\_PI)\*sin(3\*M\_PI\*(coords\_X->data[i]+coords\_Y->data[i]));

}

}

/\* Ініціалізуємо умови локальної зупинки алгоритму\*/

local\_stop=(bool\*)malloc(points\_per\_node\*points\_per\_node\*sizeof(bool));

for(int i=0;i<points\_per\_node\*points\_per\_node;i++){

local\_stop[i]=false;

}

/\* Створимо масив значень функції в локальних узлах сітки дискретизації\*/

f\_xy=matrix\_alloc(points\_per\_node,points\_per\_node,0.0);

for(int i=0;i<points\_per\_node\*points\_per\_node;i++){

f\_xy->data[i]=func(coords\_X->data[i],coords\_Y->data[i]);

}

/\*Визначимо типи для передач під час ітерацій\*/

/\* Тип, що визначає стовпець матриці\*/

MPI\_Type\_vector(points\_per\_node,1,points\_per\_node,MPI\_DOUBLE,&mat\_col);

MPI\_Type\_commit(&mat\_col);/\* Регіструємо тип \*/

left\_col=(double\*)malloc(points\_per\_node\*points\_per\_node\*sizeof(double));

right\_col=(double\*)malloc(points\_per\_node\*points\_per\_node\*sizeof(double));

top\_row=(double\*)malloc(points\_per\_node\*points\_per\_node\*sizeof(double));

bot\_row=(double\*)malloc(points\_per\_node\*points\_per\_node\*sizeof(double));

/\* Початок ітерацій \*/

do{

for(int i=1;i<points\_per\_node-1;i++){/\*Внутрішні точки\*/

for(int j=1;j<points\_per\_node-1;i++){

if (!(local\_stop[i\*points\_per\_node+j])){

int old\_node=node\_mat->data[i\*points\_per\_node+j];

double LU=old\_node-(node\_mat->data[i\*points\_per\_node+j-1]

+node\_mat->data[i\*points\_per\_node+j+1]+

node\_mat->data[(i-1)\*points\_per\_node+j]+

node\_mat->data[(i+1)\*points\_per\_node+j]-4\*old\_node)/(h\*h);

node\_mat->data[i\*points\_per\_node+j]=old\_node-(omega\*h\*h/4)\*

(LU-f\_xy->data[i\*points\_per\_node+j]);

if ((node\_mat->data[i\*points\_per\_node+j]-old\_node)

/node\_mat->data[i\*points\_per\_node+j]<epsilon){

local\_stop[i\*points\_per\_node+j]=true;

}

}

}

}

/\* Обмін між процессами у сітці\*/

if(node\_X!=0){

/\* Приймемо стовпець з лівого процессу \*/

MPI\_Recv(left\_col,1,mat\_col,rank-1,rank-1,MPI\_COMM\_WORLD,&stat);

}

if(node\_X!=nodes\_width-1){

/\* Відішлемо стовпець до правого процессу \*/

MPI\_Send(node\_mat->data+points\_per\_node-1,1,mat\_col

,rank+1,rank,MPI\_COMM\_WORLD);

/\* Приймемо стовпець з правого процессу \*/

MPI\_Recv(right\_col,1,mat\_col,rank+1,rank+1,MPI\_COMM\_WORLD,&stat);

}

if(node\_X!=0){

/\* Відішлемо стовпець до лівого процессу \*/

MPI\_Send(node\_mat->data,1,mat\_col,rank-1,rank,MPI\_COMM\_WORLD);

}

if(node\_Y!=0){

/\* Приймемо строку з верхнього процессу \*/

MPI\_Recv(top\_row,points\_per\_node,MPI\_DOUBLE,rank-nodes\_width

,rank-nodes\_width,MPI\_COMM\_WORLD,&stat);

}

if(node\_Y!=nodes\_width-1){

/\* Відішлемо строку до нижнього процессу \*/

MPI\_Send(node\_mat->data+(points\_per\_node-1)\*points\_per\_node,points\_per\_node

,MPI\_DOUBLE,rank+nodes\_width,rank,MPI\_COMM\_WORLD);

/\* Приймемо строку з нижнього процессу \*/

MPI\_Recv(bot\_row,points\_per\_node,MPI\_DOUBLE,rank+nodes\_width,

rank+nodes\_width,MPI\_COMM\_WORLD,&stat);

}

if(node\_Y!=0){

/\* Відішлемо строку до верхнього процессу \*/

MPI\_Send(node\_mat->data,points\_per\_node,MPI\_DOUBLE,

rank-nodes\_width,rank,MPI\_COMM\_WORLD);

}

/\* Граничні точки процесса \*/

if(!(node\_X==0||node\_Y==0)){/\* Лівий верхній кут \*/

if (!(local\_stop[0])){

int old\_node=node\_mat->data[0];

double LU=old\_node-(left\_col[0]

+top\_row[0]+

node\_mat->data[points\_per\_node]+

node\_mat->data[1]-4\*old\_node)/(h\*h);

node\_mat->data[0]=old\_node-(omega\*h\*h/4)\*(LU-f\_xy->data[0]);

if ((node\_mat->data[0]-old\_node)/node\_mat->data[0]<epsilon){

local\_stop[0]=true;

}

}

}

if(!(node\_X==nodes\_width-1||node\_Y==0)){/\* Правий верхній кут \*/

if (!(local\_stop[points\_per\_node-1])){

int old\_node=node\_mat->data[points\_per\_node-1];

double LU=old\_node-(right\_col[0]

+top\_row[points\_per\_node-1]+

node\_mat->data[2\*points\_per\_node-1]+

node\_mat->data[points\_per\_node-2]-4\*old\_node)/(h\*h);

node\_mat->data[points\_per\_node-1]=old\_node-(omega\*h\*h/4)

\*(LU-f\_xy->data[points\_per\_node-1]);

if ((node\_mat->data[points\_per\_node-1]-old\_node)

/node\_mat->data[points\_per\_node-1]<epsilon){

local\_stop[points\_per\_node-1]=true;

}

}

}

if(!(node\_X==0||node\_Y==nodes\_width-1)){/\* Лівий нижній кут \*/

//printf("%d %d \n",node\_X,node\_Y);

if (!(local\_stop[(points\_per\_node-1)\*points\_per\_node])){

int old\_node=node\_mat->data[(points\_per\_node-1)\*points\_per\_node];

double LU=old\_node-(left\_col[points\_per\_node\*(points\_per\_node-1)]

+bot\_row[0]+

node\_mat->data[(points\_per\_node-1)\*points\_per\_node+1]+

node\_mat->data[(points\_per\_node-2)\*points\_per\_node]-4\*old\_node)/(h\*h);

node\_mat->data[(points\_per\_node-1)\*points\_per\_node]=old\_node-

(omega\*h\*h/4)\*(LU-f\_xy->data[(points\_per\_node-1)\*points\_per\_node]);

if ((node\_mat->data[(points\_per\_node-1)\*points\_per\_node]-old\_node)

/node\_mat->data[(points\_per\_node-1)\*points\_per\_node]<epsilon){

local\_stop[(points\_per\_node-1)\*points\_per\_node]=true;

}

}

}

if(!(node\_X==nodes\_width-1||node\_Y==nodes\_width-1)){/\* Правий нижній кут \*/

if (!(local\_stop[points\_per\_node\*points\_per\_node-1])){

int old\_node=node\_mat->data[points\_per\_node\*points\_per\_node-1];

double LU=old\_node-(right\_col[points\_per\_node\*(points\_per\_node-1)]

+bot\_row[points\_per\_node-1]+

node\_mat->data[points\_per\_node\*points\_per\_node-2]+

node\_mat->data[(points\_per\_node-1)\*points\_per\_node-1]-4\*old\_node)/(h\*h);

node\_mat->data[points\_per\_node\*points\_per\_node-1]=old\_node-

(omega\*h\*h/4)\*(LU-f\_xy->data[points\_per\_node\*points\_per\_node-1]);

if ((node\_mat->data[points\_per\_node\*points\_per\_node-1]-old\_node)

/node\_mat->data[points\_per\_node\*points\_per\_node-1]<epsilon){

local\_stop[points\_per\_node\*points\_per\_node-1]=true;

}

}

}

if(node\_X!=0){/\* Ліва сторона \*/

for(int i=1;i<points\_per\_node-1;i++){

if (!(local\_stop[i\*points\_per\_node])){

int old\_node=node\_mat->data[i\*points\_per\_node];

double LU=old\_node-(node\_mat->data[i\*points\_per\_node+1]

+node\_mat->data[(i-1)\*points\_per\_node]+

left\_col[i]+

node\_mat->data[(i+1)\*points\_per\_node]-4\*old\_node)/(h\*h);

node\_mat->data[i\*points\_per\_node]=old\_node-

(omega\*h\*h/4)\*(LU-f\_xy->data[i\*points\_per\_node]);

if ((node\_mat->data[i\*points\_per\_node]-old\_node)

/node\_mat->data[i\*points\_per\_node]<epsilon){

local\_stop[i\*points\_per\_node]=true;

}

}

}

}

if(node\_X!=nodes\_width-1){/\* Права сторона \*/

for(int i=1;i<points\_per\_node-1;i++){

if (!(local\_stop[(i+1)\*points\_per\_node-1])){

int old\_node=node\_mat->data[(i+1)\*points\_per\_node-1];

double LU=old\_node-(node\_mat->data[(i+2)\*points\_per\_node-1]

+node\_mat->data[(i)\*points\_per\_node-1]+

right\_col[i]+

node\_mat->data[(i+1)\*points\_per\_node-2]-4\*old\_node)/(h\*h);

node\_mat->data[i\*points\_per\_node]=old\_node-

(omega\*h\*h/4)\*(LU-f\_xy->data[(i+1)\*points\_per\_node-1]);

if ((node\_mat->data[(i+1)\*points\_per\_node-1]-old\_node)

/node\_mat->data[(i+1)\*points\_per\_node-1]<epsilon){

local\_stop[(i+1)\*points\_per\_node-1]=true;

}

}

}

}

if(node\_Y!=0){/\* Верхня сторона \*/

for(int i=1;i<points\_per\_node-1;i++){

if (!(local\_stop[i])){

int old\_node=node\_mat->data[i];

double LU=old\_node-(node\_mat->data[i-1]

+node\_mat->data[i+1]+

top\_row[i]+

node\_mat->data[points\_per\_node+i]-4\*old\_node)/(h\*h);

node\_mat->data[i]=old\_node-(omega\*h\*h/4)\*(LU-f\_xy->data[i]);

if ((node\_mat->data[i]-old\_node)/node\_mat->data[i]<epsilon){

local\_stop[i]=true;

}

}

}

}

if(node\_Y!=nodes\_width-1){/\* Нижня сторона \*/

for(int i=1;i<points\_per\_node-1;i++){

if (!(local\_stop[(points\_per\_node-1)\*points\_per\_node+i])){

int old\_node=node\_mat->data[(points\_per\_node-1)\*points\_per\_node+i];

double LU=old\_node-(node\_mat->data[(points\_per\_node-1)\*points\_per\_node+i-1]

+node\_mat->data[(points\_per\_node-1)\*points\_per\_node+i+1]+

bot\_row[i]+

node\_mat->data[(points\_per\_node-2)\*points\_per\_node+i]-4\*old\_node)/(h\*h);

node\_mat->data[(points\_per\_node-1)\*points\_per\_node+i]=old\_node-

(omega\*h\*h/4)\*(LU-f\_xy->data[(points\_per\_node-1)\*points\_per\_node+i]);

if ((node\_mat->data[(points\_per\_node-1)\*points\_per\_node+i]-old\_node)

/node\_mat->data[(points\_per\_node-1)\*points\_per\_node+i]<epsilon){

local\_stop[(points\_per\_node-1)\*points\_per\_node+i]=true;

}

}

}

}

/\*Якщо усі вузли сітки дискретизації у процесі завершили

обчислення - встановлюємо ознаку завершення обчислень у процесі\*/

node\_stop=true;

for(int i=0;i<points\_per\_node\*points\_per\_node;i++){

if (!local\_stop[i]){

node\_stop=false;

break;

}

}

global\_stop=true;

/\*Перевіряємо чи всі процеси закінчили обчислення\*/

MPI\_Reduce(&node\_stop,&global\_stop,1,MPI\_SHORT,MPI\_LAND,0,MPI\_COMM\_WORLD);

/\*Якщо всі - то посилаємо сигнал про завершення\*/

MPI\_Bcast(&global\_stop,1,MPI\_BYTE,0,MPI\_COMM\_WORLD);

} while(!global\_stop);

/\* Збираємо результат \*/

total\_mat=matrix\_alloc(total\_width,total\_width,0.0);

if(rank==0){

for(int i=0;i<nodes\_width;i++){

for(int j=0;j<nodes\_width;j++){

if(!(i==0&&j==0)){

my\_matrix\* temp=matrix\_alloc(points\_per\_node,points\_per\_node,0.0);

/\* Приймаємо частини матриці сітки дискретизації з кожного процессу \*/

MPI\_Recv(temp->data,points\_per\_node\*points\_per\_node,

MPI\_DOUBLE,i\*nodes\_width+j,i\*nodes\_width+j,MPI\_COMM\_WORLD,&stat);

/\* Та розташовуємо їх у повній матриці \*/

for(int k=0;k<points\_per\_node;k++){

for(int l=0;l<points\_per\_node;l++){

total\_mat->data[i\*total\_width\*points\_per\_node+

j\*points\_per\_node+k\*total\_width+l]

=temp->data[k\*points\_per\_node+l];

}

}

}

}

}

for(int i=0;i<points\_per\_node;i++){

for(int j=0;j<points\_per\_node;j++){

total\_mat->data[i\*total\_width+j]=node\_mat->data[i\*points\_per\_node+j];

}

}

}

else{

MPI\_Send(node\_mat->data,points\_per\_node\*points\_per\_node

,MPI\_DOUBLE,0,rank,MPI\_COMM\_WORLD);

}

if (rank==0){

matrix\_print(outpfile,total\_mat);;

}

MPI\_Finalize();

}

/\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*/

/\* Допоміжні функції \*/

double func(double X,double Y){

return X\*Y;

}

void fatal\_error(const char \*message, int errorcode)

{

printf("fatal error: code %d, %s\n", errorcode, message);

fflush(stdout);

MPI\_Abort(MPI\_COMM\_WORLD, errorcode);

}

struct my\_matrix \*matrix\_alloc(int rows, int cols, double initial)

{

struct my\_matrix \*result = (my\_matrix\*)malloc(sizeof(struct my\_matrix));

result->rows = rows;

result->cols = cols;

result->data = (double\*)malloc(sizeof(double) \* rows \* cols);

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

{

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

{

result->data[i \* cols + j] = initial;

}

}

return result;

}

void matrix\_print(const char \*filename, struct my\_matrix \*mat)

{

FILE \*f = fopen(filename, "w");

if(f == NULL)

{

fatal\_error("cant write to file", 2);

}

for(int i = 0; i < mat->rows; i++)

{

for(int j = 0; j < mat->cols; j++)

{

fprintf(f, "%lf ", mat->data[i \* mat->cols + j]);

}

fprintf(f, "\n");

}

fclose(f);

}

struct my\_matrix \*read\_matrix(const char \*filename)

{

FILE \*mat\_file = fopen(filename, "r");

if(mat\_file == NULL)

{

fatal\_error("can't open matrix file", 1);

}

int rows;

int cols;

fscanf(mat\_file, "%d %d", &rows, &cols);

struct my\_matrix \*result = matrix\_alloc(rows, cols, 0.0);

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

{

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

{

fscanf(mat\_file, "%lf", &result->data[i \* cols + j]);

}

}

fclose(mat\_file);

return result;

}