Лабораторна робота №4

Знайомство з технологією паралельного програмування MPI

Мета: Вивчити основи програмування і виробити практичні навики роботи з технологією MPI.

1.1. Поняття паралельної програми

Під паралельною програмою в рамках MPI розуміється безліч одночасно виконуваних процесів. Процеси можуть виконуватися на різних процесорах, але на одному процесорі можуть розташовуватися і декілька процесів (в цьому випадку їх виконання здійснюється в режимі розділення часу). У граничному випадку для виконання паралельної програми може використовуватися один процесор – як правило, такий спосіб застосовується для початкової перевірки правильності паралельної програми.

Кількість процесів і число використовуваних процесорів визначається у момент запуску паралельної програми засобами середовища виконання mpi-програм і в ході обчислень мінятися не може (у стандарті Mpi-2 передбачається можливість динамічної зміни кількості процесів). Всі процеси програми послідовно перенумеровані від **0** до **p-1**, де **p** є загальна кількість процесів. Номер процесу іменується рангом процесу.

1.2. Ініціалізація і завершення MPI програм

Першою функцією MPI, що викликається, має бути функція:

int **MPI_Init** (int *agrc, char ***argv);

Параметрами функції є кількість аргументів в командній строчці і текст самої командної строчки.

Останньою функцією MPI, що викликається, обов'язково має бути функція:

int **MPI_Finalize** (void);

Як результат, можна відзначити, що структура паралельної програми, розроблена з використанням MPI, повинна мати наступний вигляд:

```
#include "mpi.h"
int main ( int argc, char *argv[] )
{
  < програмний код без використання MPI функцій >
  MPI_Init ( &agrc, &argv );
    < програмний код з використанням MPI функцій >
  MPI_Finalize();
  < програмний код без використання MPI функцій >
  return 0;
}
```

1.2 Визначення кількості і рангу процесів

Визначення кількості процесів у виконуваній паралельній програмі здійснюється за допомогою функції:

int **MPI_Comm_size** (MPI_Comm comm, int *size).

Для визначення рангу процесу використовується функція:

int **MPI_Comm_rank** (MPI_Comm comm, int *rank).

Як правило, виклик функцій **MPI_Comm_size** и **MPI_Comm_rank** виконується відразу після **MPI_Init**:

```
#include "mpi.h"
int main ( int argc, char *argv[] )
{
  int ProcNum, ProcRank;
  < програмний код без використання MPI функцій >
  MPI_Init ( &agrc, &argv );
  MPI_Comm_size ( MPI_COMM_WORLD, &ProcNum);
  MPI_Comm_rank ( MPI_COMM_WORLD, &ProcRank);
    < програмний код з використанням MPI функцій >
  MPI_Finalize();
  < програмний код без використання MPI функцій >
  return 0;
}
```

Слід зазначити:

1. Коммунікатор **MPI_COMM_WORLD** створюється за умовчанням і представляє всі процеси виконуваної паралельної програми;
2. Ранг, що отримується за допомогою функції **MPI_Comm_rank**, є рангом процесу, що виконав виклик цієї функції, тобто змінна **ProcRank** набуватиме різних значень в різних процесах.

1.3. Передача повідомлень

Для передачі повідомлення процес-відправник повинні виконати функцію:

int **MPI_Send**(void *buf, int count, MPI_Datatype type, int dest,
int tag, MPI_Comm comm),
де:

- **buf** - адреса буфера пам'яті, в якому розташовуються дані повідомлення, що відправляється,
- **count** - кількість елементів даних в повідомленні,
- **type** - тип елементів даних повідомлення, що пересилається,
- **dest** - ранг процесу, якому вирушає повідомлення,
- **tag** - значення-тег, використовуване для ідентифікації повідомленя,

- **comm** - комунікатор, в рамках якого виконується передача даних.

Для вказівки типа даних, що пересилаються, в MPI є ряд базових типів, повний список яких приведений в табл. 1.

Таблиця 1. Базові типи даних MPI для алгоритмічної мови C

MPI_DATATYPE	C DATATYPY
MPI_BYTE	
MPI_CHAR	Signed char
MPI_DOUBLE	Double
MPI_FLOAT	Float
MPI_INT	Int
MPI_LONG	Long
MPI_LONG_DOUBLE	Long double
MPI_PACKED	
MPI_SHORT	Short
MPI_UNSIGNED_CHAR	Unsigned char
MPI_UNSIGNED	Unsigned int
MPI_UNSIGNED_LONG	Unsigned long
MPI_UNSIGNED_SHORT	Unsigned short

1.4. Прийом повідомлень

Для прийому повідомлення процес-одержувач повинен виконати функцію:

int **MPI_Recv**(void *buf, int count, MPI_Datatype type, int source,
int tag, MPI_Comm comm, MPI_Status *status),
де:

- **buf, count, type** – буфер пам'яті для прийому повідомлення, призначення кожного окремого параметра відповідає опису в MPI_Send,
- **source** - ранг процесу, від якого має бути виконаний прийом повідомлення,
- **tag** - тег повідомлення, яке має бути прийняте для процесу,
- **comm** - комунікатор, в рамках якого виконується передача даних,
- **status** – покажчик на структуру даних з інформацією про результат виконання операції прийому даних.

Виклик функції **MPI_Recv** не повинен узгоджуватися з часом виклику відповідної функції передачі повідомлення **MPI_Send** – прийом повідомлення може бути ініційований до моменту, в момент або після моменту початку відправки повідомлення.

Після закінчення роботи функції **MPI_Recv** у заданому буфері пам'яті розташовуватиметься прийняте повідомлення. Принциповий момент тут полягає в тому, що функція **MPI_Recv** є блокуючою для процесу-одержувача, тобто його виконання припиняється до завершення роботи функції. Таким чином, якщо по якихось причинах очікуване для прийому повідомлення буде відсутнє, виконання паралельної програми буде блоковано.

Розглянутий набір функцій виявляється достатнім для розробки паралельних програм.

1.5. Приклад паралельної програми з використанням MPI

Програма, що приводиться нижче, є стандартним початковим прикладом для алгоритмічної мови С.

```
#include <stdio.h>
#include "mpi.h"
int main(int argc, char* argv[])
{
  int ProcNum, ProcRank, RecvRank;
  MPI_Status Status;
  MPI_Init(&argc, &argv);
  MPI_Comm_size(MPI_COMM_WORLD, &ProcNum);
  MPI_Comm_rank(MPI_COMM_WORLD, &ProcRank);
  if ( ProcRank == 0 )
  {
    printf ("\n Hello from process %3d", ProcRank);
    for ( int i=1; i<ProcNum; i++ )
    {
      MPI_Recv(&RecvRank, 1, MPI_INT, MPI_ANY_SOURCE,
        MPI_ANY_TAG, MPI_COMM_WORLD, &Status);
      printf("\n Hello from process %3d", RecvRank);
    }
  }
  else
    MPI_Send(&ProcRank,1,MPI_INT,0,0,MPI_COMM_WORLD);
  MPI_Finalize();
  return 0;
}
```

Як випливає з тексту програми, кожен процес визначає свій ранг, після чого дії в програмі розділяються. Всі процеси, окрім процесу з рангом 0, передають значення свого рангу нульовому процесу. Процес з рангом 0 спочатку друкує значення свого рангу, а далі послідовно приймає повідомлення з рангами процесів і також друкує їх значення. При цьому важно відзначити, що порядок прийому повідомлень заздалегідь не визначений і залежить від умов виконання паралельної програми (більш того, цей порядок може змінюватися від запуску до запуску). Так, можливий варіант результатів друку процесу 0 може полягати в наступному (для паралельної програми з чотирьох процесів):

Hello from process 0
Hello from process 2

Hello from process 1
Hello from process 3

Такий "плаваючий" вигляд отримуваних результатів істотним чином ускладнює розробку, тестування і відлагодження паралельних програм, оскільки в цьому випадку зникає один з основних принципів програмування – повторюваність виконуваних обчислювальних експериментів. Як правило, якщо це не приводить до втрати ефективності, слід забезпечувати однозначність розрахунків і при використанні паралельних обчислень. Так, для даного простого прикладу можна відновити постійність отримуваних результатів за допомогою завдання рангу процесу-відправника в операції прийому повідомлення:

MPI_Recv(&RecvRank, 1, MPI_INT, **i,** MPI_ANY_TAG, MPI_COMM_WORLD, &Status).

Вказівка рангу процесу-відправника регламентує порядок прийому повідомлень, і, як результат, рядки друку з'являтимуться строго в порядку зростання рангів процесів (повторимо, що така регламентація в окремих ситуаціях може приводити до уповільнення виконуваних паралельних обчислень).

Ділянка з функцією прийому **Mpi_recv** виконується лише процесом з рангом 0, ділянка з функцією передачі **Mpi_ send** використовується всіма процесами, за винятком нульового процесу.

Для розділення фрагментів коду між процесами зазвичай використовується підхід, застосований в тільки що розглянутій програмі, - за допомогою функції **Mpi_comm_rank** визначається ранг процесу, а потім відповідно до рангу виділяються необхідні для процесу ділянки програмної коду. Наявність в одній і тій же програмі фрагментів коду різних процесів також значно ускладнює розуміння і, в цілому, розробку mpi-програм. Як результат, можна рекомендувати при збільшенні об'єму програм, що розробляються, виносити програмний код різних процесів в окремі програмні модулі (функції). Загальна схема MPI програми в цьому випадку матиме вигляд::

```
MPI_Comm_rank(MPI_COMM_WORLD, &ProcRank);
if ( ProcRank == 0 ) DoProcess0();
else if ( ProcRank == 1 ) DoProcess1();
else if ( ProcRank == 2 ) DoProcess2();
```

У багатьох випадках, як і в розглянутому прикладі, виконувані дії є такими, що відрізняються лише для процесу з рангом 0. В цьому випадку загальна схема MPI програми приймає простіший вигляд:

```
MPI_Comm_rank(MPI_COMM_WORLD, &ProcRank);
if ( ProcRank == 0 ) DoManagerProcess();
else DoWorkerProcesses();
```

Хід роботи

Створити програму яка повинна реалізувати наступні дії:

1.Створити матрицю A розміром m * n, елементи якої заповнюються рандомно, m задає кількість рядків, n задає кількість стовпців.

2. У процесі з рангом 0 вивести наступні дані: номер лабораторної роботи; назва лабораторної роботи; групу студента; ФІО студента; номер варіанту завдання.

3. Виконати розсилку частин даних матриці всім процесам паралельної області. Обробити паралельним способом матрицю відповідно до свого варіанту.

4. Результати обробки вивести на екран, вказавши ранг процесу, який обробляв відповідний рядок матриці.

5. Виконати глобальну операцію редукції по всій матриці за вказаним варіантом і вивести отримане значення у процесі з рангом 0.

№ Варіанта	Завдання
1	Знайти мінімальне значення кожного рядка матриці
2	Знайти максимальне значення кожного рядка матриці
3	Порахувати кількість додатних елементів в кожному рядку матриці
4	Порахувати кількість від’ємних елементів в кожному рядку матриці
5	Порахувати кількість нульових елементів в кожному рядку матриці
6	Порахувати суму елементів в кожному рядку матриці
7	Порахувати суму додатних елементів в кожному рядку матриці
8	Порахувати суму від’ємних елементів в кожному рядку матриці

Контрольні питання:

1. Що слід розуміти під паралельною програмою?

2. У чому відмінність понять процесу і процесора?

3. Який мінімальний набір функцій MPI дозволяє почати розробку паралельних програм?

4. Як описуються повідомлення, що передаються?

5. Як можна організувати прийом повідомлень від конкретних процесів?

6. Які режими обміну повідомленнями існують в MPI?

7. Чим відрізняються операції блокуючого обміну повідомленнями від неблокуючого?

8. У яких випадках при обміні повідомленнями доцільно використовувати значення MPI_ANY_SOURCE та MPI_ANY_TAG.