Лекція 6. Технологія передачі повідомлень MPI

План лекції:

1. Введення в паралельне програмування з використанням MPI
2. Операції обміну повідомленнями
3. MPI — Інтерфейс Передачі Повідомлень
4. Коди завершення
5. Як влаштована MPI-програма

Введення в паралельне програмування з використанням MPI

У моделі передачі повідомлень, яка є різновидом паралельної моделі програмування, архітектура комп'ютера відрізняється від фон-нейманівскої. У цьому випадку вважають, що комп'ютер складається з декількох процесорів, кожен з яких забезпечений своєю власною пам'яттю. Процесор та його власна оперативна пам'ять, тобто комп'ютер, включений в комунікаційну мережу, яку часто називають хост-машиною (від англійського host) або, на жаргоні програмістів, просто "хост". Паралельна програма в моделі передачі повідомлень являє собою набір звичайних послідовних програм, які відпрацьовуються одночасно. Звичайно, кожна з цих послідовних програм виконується на своєму процесорі і має доступ до своєї локальної пам'яті. Очевидно, в такому разі потрібен механізм, що забезпечує узгоджену роботу частин паралельної програми. Частини паралельної програми в процесі їх виконання будемо називати підзадачами.

Для забезпечення узгодженої роботи підзадачі повинні обмінюватися між собою інформацією. Інформація може бути різною. Це можуть бути дані, на обробку яких виконує програма, а можуть бути і керуючі сигнали. У моделі передачі повідомлень пересилання даних і керуючих сигналів відбуваються за допомогою повідомлень. Звернемо увагу на те, що обмін відбувається не через загальну або поділювану пам'ять, а через інші комунікаційні середовища, тому дана модель орієнтована на обчислювальні системи з розподіленою пам'яттю. Пересилання повідомлень — їх відправка і прийом реалізуються програмістом за допомогою виклику відповідних підпрограм бібліотеки передачі повідомлень.

Модель передачі повідомлень універсальна. Вона може бути реалізована на паралельних обчислювальних системах як розподілених, так і з пам'яттю, що розділяється, на кластерах робочих станцій і навіть на звичайних однопроцесорних комп'ютерах. В останньому випадку, паралельна програма зазвичай виконується в режимі налагодження.

Універсальність і незалежність від архітектури, прихованої від програміста, стали однією з основних причин популярності моделі передачі повідомлень.

При розробці паралельного алгоритму, на етапі декомпозиції вихідна проблема розбивається на декілька частин-підзадач, кожна з яких часто може бути вирішена одним і тим же методом. Загальний метод застосовується до різних фрагментів набору даних, що підлягає обробці. У цьому випадку паралельна програма складається з однакових фрагментів і на всіх процесорах виконуються однакові підзадачі, фактично, одна і та ж програма. Вона може запускатися виділеною підзадачею, що грає роль "диригента" (ми будемо надалі називати її майстер-програмою), або іншими підзадачами, що входять в паралельне застосування. Така схема називається SPMD-моделлю (Single Program Multiple Data) і є окремим випадком моделі передачі повідомлень. У деяких програмних реалізаціях моделі використовується саме SPMD-варіант, а запуск декількох підзадач на одному процесорі заборонений.

Повідомлення містить дані, що пересилаються і службову інформацію:

- ідентифікатор процесу-відправника повідомлення. У MPI ідентифікатор процесу називають рангом;
- адреса, за якою розміщуються дані, що пересилаються процесом-відправником;
- тип даних, що пересилаються;
- кількість даних (розмір буфера повідомлення — для того, щоб прийняти повідомлення, процес повинен відвести для нього достатній обсяг оперативної пам'яті!);
- ідентифікатор процесу, який повинен отримати повідомлення;
- адресу, за якою повинні бути розміщені дані процесом-одержувачем.

Частина службової інформації становить "конверт" повідомлення. В "конверті" містяться:

- ранг джерела;
- ранг адресата;
- тег повідомлення;
- ідентифікатор комунікатора, що описує область взаємодії усередині якої відбувається обмін.

Ці дані дозволяють адресату розрізняти повідомлення.

Ранг джерела дає можливість розрізняти повідомлення, що приходять від різних процесів.

Тег - це задається користувачем ціле число від 0 до 32 767, яке відіграє роль ідентифікатора повідомлення і дозволяє розрізняти повідомлення, що приходять від одного процесу. Теги можуть використовуватися і для дотримання певного порядку прийому повідомлень.

Дані, що містяться у повідомленні, у загальному випадку організовані в масив елементів, кожен з яких має певний тип.

Крім пересилання даних система передачі повідомлень повинна підтримувати пересилку інформації про стан процесів комунікації. Це може бути, наприклад, повідомлення про те, що прийом даних, відправлених іншим процесом, завершено.

Перед використанням процедур передачі повідомлень програма повинна "підключитися" до системи обміну повідомленнями. Підключення виконується за допомогою відповідного виклику процедури з бібліотеки. В одних реалізаціях моделі допускається тільки одне підключення, а в інших декілька підключень до системи.

Прийом повідомлення починається з підготовки буфера достатнього розміру. В цей буфер записуються дані, що приймаються. Дані можуть бути закодовані, в цьому випадку при їх зчитуванні виконується декодування. Кодування даних, тобто їх перетворення в деякий універсальний формат, що потрібно для того, щоб забезпечити обмін повідомленнями між системами з різним форматом представлення даних.

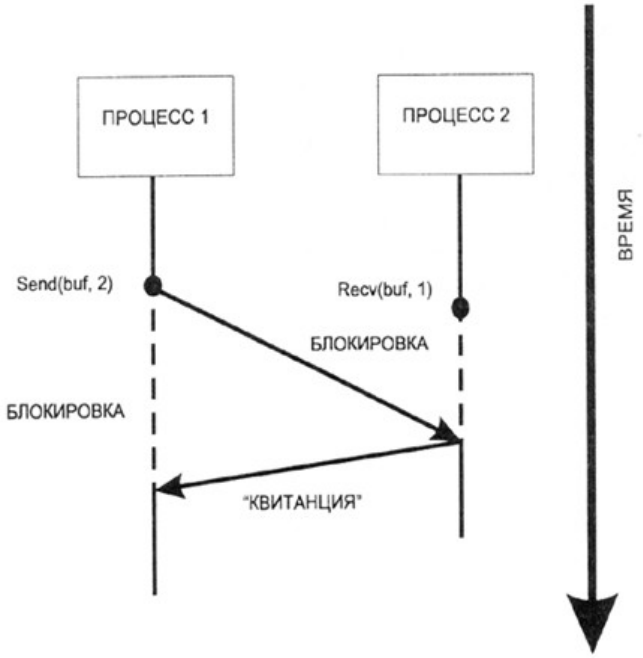
Операція відправлення або отримання повідомлення вважається завершеною, якщо програма може знову використовувати такі ресурси, як буфери повідомлень.

Операції обміну повідомленнями

Двоточковий (point-to-point) обмін — це найпростіша форма обміну повідомленнями. Називається він так тому, що в ньому беруть участь тільки два процеса, процес-відправник і процес-одержувач — джерело і адресат.

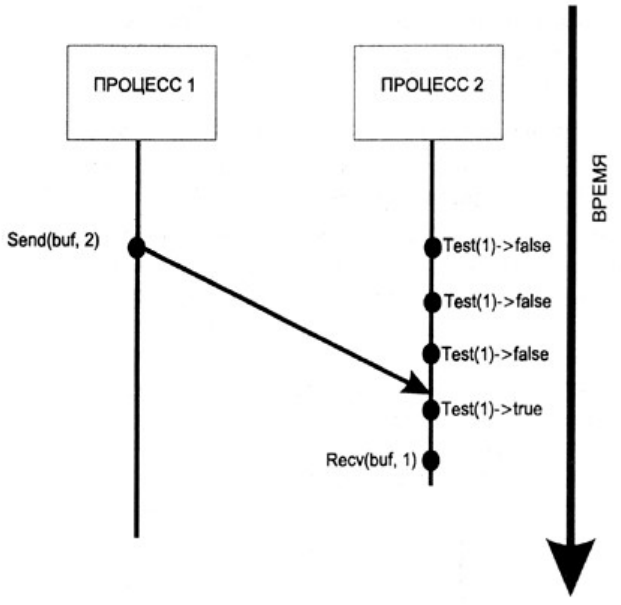
Є декілька різновидів двоточкового обміну:

- *синхронний обмін,* який супроводжується повідомленням про закінчення прийому повідомлення;
- *асинхронний обмін,* який таким повідомленням не супроводжується;
- *блокуючі прийом/передача,* які призупиняють виконання процесу на час прийому повідомлення (мал. 6.1);
- *не блокуючий прийом/передача,* при яких виконання процесу продовжується у фоновому режимі, а програма в потрібний момент може запитати підтвердження завершення прийому повідомлення.



Мал. 6.1. Блокуючі прийом/передача

Не блокуючий обмін вимагає акуратності при використанні функцій прийому. Оскільки не блокуючий прийом завершується негайно, для системи неважливо, прибуло повідомлення до місця призначення чи ні. Переконатися в цьому можна з допомогою функцій перевірки отримання повідомлення. Зазвичай виклик таких функцій розміщується в циклі, який повторюється до тих пір, поки функція перевірки не поверне значення "істина" (перевірка отримання пройшла успішно). Після цього можна викликати функцію прийому повідомлення з буфера повідомлень (мал. 6.2).



Мал. 6.2. Не блокуючий прийом/передача

В операції *колективного* обміну залучені не два, а більше число процесів. Різновидами колективного обміну є:

- широкомовна передача — виконується від одного процесу до всіх;
- обмін з бар'єром — це форма синхронізації роботи процесів, коли обмін повідомленнями відбувається тільки після того, як до відповідної процедури звернулося певне число процесів;
- операції приведення — вхідними є дані декількох процесів, а результат — одне значення, яке стає доступним всім процесам, які беруть участь в обміні.

Важлива властивість системи передачі повідомлень полягає в тому, чи гарантує вона збереження порядку прийому повідомлень. Якщо гарантує, то при відправці одним процесом іншому декількох повідомлень вони будуть отримані в тій же послідовності, в якій були відправлені. Більшість реалізацій моделі передачі повідомлень володіє даною властивістю, але не у всіх режимах обміну.

Є три способи реалізації моделі передачі повідомлень:

- створення спеціалізованої мови паралельного програмування;
- розширення звичайної послідовної мови шляхом включення в неї засобів обміну повідомленнями;
- використання спеціалізованих бібліотек в програмах, написаних на звичайних мовах послідовного програмування.

Мова Occam, розроблена для трансп’ютерних систем, є прикладом першого підходу.

Приклад другого підходу — мови CC+ (Compositional C++ — розширення C++) та FORTRAN M (розширення мови FORTRAN).

Є безліч бібліотек передачі повідомлень, як вільно розповсюджуваних, так і комерційних, призначених для конкретних платформ. Найчастіше використовуються незалежні від платформи (платформою називають поєднання архітектури комп'ютера і встановленої на ньому операційної системи) бібліотеки: PVM (Parallel Virtual Machine — "Паралельна Віртуальна Машина") і різні реалізації MPI (Message Passing Interface). Приклад платформо-залежної бібліотеки — бібліотека для обчислювальної системи nCUBE. Конкретні набори функцій в цих бібліотеках можуть відрізнятися, але базовий набір приблизно однаковий.

MPI — Інтерфейс Передачі Повідомлень

Практичне втілення модель передачі повідомлень знайшла в специфікації, яка отримала назву Інтерфейс Передачі Повідомлень MPI. Ця специфікація була розроблена в

1993-1994 роках групою MPI Forum, до складу якої входили представники академічних і промислових кіл. Вона стала першим стандартом систем передачі повідомлень. У MPI були враховані досягнення інших проектів по створенню систем передачі повідомлень: NX/2, Express, nCUBE, Vertex, p4, PARMACS, PVM, Хамелеон, Zipcode, Chimp і т. д. Її реалізації являють собою бібліотеки підпрограм, які можуть використовуватися в програмах на мовах C/C++ та FORTRAN. В даний час прийнята нова версія специфікації - MPI-2.

У моделі програмування, яку підтримує MPI, програма породжує кілька процесів взаємодіючих між собою за допомогою звернень до підпрограм передачі і прийому повідомлень. Зазвичай, при ініціалізації MPI-програми створюється фіксований набір процесів, причому кожен процес виконується на своєму процесорі. У цих процесах можуть виконуватися різні програми, тому модель програмування MPI іноді називають MPMD-моделлю (Multiple Program Multiple Data — безліч програм безліч даних), на відміну від SPMD-моделі, де на кожному процесорі виконуються тільки однакові завдання.

Двоточкові обміни використовуються для організації локальних і неструктурованих комунікацій. При виконанні глобальних операцій застосовуються колективні обміни. Асинхронні комунікації реалізуються за допомогою запитів про отримання повідомлень. Механізм, який називається комунікатором, приховує від програміста внутрішні комунікаційні структури.

Серед безлічі підпрограм MPI можна виділити 6 основних, тих, які використовуються найчастіше:

1. MPI_INIT(int *argc, char **argv) — підключення до MPI. Аргументи argc і argv потрібні тільки в тих програмах, де вони задають кількість аргументів командного рядка запуску програми і вектор цих аргументів. Цей виклик передує всім іншим викликів підпрограм MPI.

2. MPI_FINALIZE ( ) — завершення роботи з MPI. Після виклику цієї підпрограми не можна викликати підпрограми MPI. MPI_FINALISE повинні викликати всі процеси перед завершенням своєї роботи.

3. MPI_COMM_SIZE(comm, size) — визначення розміру області взаємодії. Тут comm — вхідний параметр-комунікатор, а вихідним є параметр size цілого типу, кількість процесів в області взаємодії.

4. MPI_COMM_RANK(comm, pid) — визначення номера процесу. Тут pid -ідентифікатор процесу у зазначеній сфері взаємодії.

5. MPI_SEND(buf, count, datatype, dest, tag, comm) — надсилання повідомлення.

Всі параметри є вхідними: buf — адреса буфера відправки, count — кількість пересилаються елементів даних (невід'ємне ціле значення), datatype — тип даних, що пересилаються, dest — ідентифікатор процесу, якому надсилаємо повідомлення, tag — тег повідомлення (ціле число), comm — комунікатор.

6. MPI_RECV(buf, count, datatype, source, tag, comm, status) — Прийом повідомлення. Параметр buf вихідний, це адреса буфера отримання даних, status теж вихідний параметр — статус завершення операції, source — ідентифікатор процесу, від якого отримуємо повідомлення (вхідний параметр). Всі інші параметри — вхідні та їх призначення збігається з призначенням аналогічних параметрів MPI_SEND.

Специфікатор source функції MPI_RECV дозволяє програмісту вказати, що повідомлення повинно бути отримано лише від конкретного процесу, заданого його цілочисельним ідентифікатором, або від будь-якого процесу. В останньому випадку використовується спеціальне значення MPI_ANY_ SOURCE. Перший спосіб краще, оскільки він виключає помилки, пов'язані з невизначеністю порядку надходження даних.

Область взаємодії (область зв'язку) визначає групу процесів. Всі процеси, що належать одній галузі взаємодії, можуть обмінюватися повідомленнями. Описує цю безліч процесів спеціальна інформаційна структура, яка називається комунікатором.

Процеси, пов'язані з MPI-програми, можуть взаємодіяти, тільки якщо вони пов'язані з одним комунікатором. Значення комунікатора (вона використовується за замовчуванням) MPI_COMM_WORLD відповідає всім процесам даної програми. Всім процесам у сфері взаємодії присвоюються цілі позитивні номери від 0 до деякого максимального, а номер поточного процесу можна визначити за допомогою виклику MPI_COMM_RANK.

З процесів, що входять в існуючу область взаємодії, можуть створюватися нові області взаємодії. Нумерація процесів у різних сферах взаємодії незалежна. Комунікатор описує область взаємодії. Для однієї галузі можуть існувати кілька комунікаторів. Стандартний комунікатор MPI_COMM_WORLD створюється автоматично.

Комунікатори є константами типу MPI_Comm в програмах на мові C і типу INTEGER у мові FORTRAN. Крім згадуваного MPI_COMM_WORLD є також: MPI_COMM_SELF — комунікатор, який містить тільки процес, що викликає сам себе, і MPI_COMM_NULL — порожній комунікатор.

Номер процесу називається його рангом. Ранг використовується для вказівки конкретного процесу, наприклад, при пересиланні повідомлень. Взагалі кажучи, процес може належати кільком областям взаємодії і в кожній у нього буде свій власний ранг.

При використанні MPI в програмах на мові С в іменах функцій використовується префікс MPI_, а перша буква імені набирається у верхньому регістрі. Не рекомендується використовувати імена, які починаються з MPI_ для своїх функцій або змінних.

В С++ підпрограма є методом для певного класу, ім'я має в цьому випадку вид MPI: :Клас: :дія_підмножина. Для деяких дій введені стандартні найменування: Create — створення нового об'єкта, Get — отримання інформації про об'єкт, set — установка параметрів об'єкта, Delete — видалення інформації, is — запит про те, чи має об'єкт зазначену властивість.

Значення коду завершення мають цілий тип і визначаються за значенням функції. Імена констант MPI записуються у верхньому регістрі. Їх опис знаходяться у файлі mpi.h який є заголовковим, який обов'язково включається в MPI-програму. Ім'я файлу може бути іншим у інших реалізаціях MPI. Вхідні параметри функцій передаються за значенням, а вихідні (і INOUT) — за посиланням. Змінна status має тип MPI_status і є структурою з полями status.MPI_SOURCE і status.MPI_TAG. У MPI прийнята своя система позначення типів даних, яка відповідає наступним типам даних в мові С.

Таблиця 6.1. Типи даних MPI для мови С

Тип даних MPI	**Тип даних С**
MPI_CHAR	signed char
MPI_SHORT	signed short int
MPI_INT	signed int
MPI_LONG	signed long int
MPI_UNSIGNED CHAR	unsigned char
MPI_UNSIGNED SHORT	unsigned short int
MPI_UNSIGNED	unsigned int
MPI_UNSIGNED LONG	unsigned long int

MPI_FLOAT	float
MPI_DOUBLE	double
MPI_LONG_DOUBLE	long double
MPI_BYTE	Немає відповідності
MPI_PACKED	Немає відповідності

У MPI повинні дотримуватися правила сумісності типів. Відповідність типів має, як правило, місце у процедурах відправки і процедурах прийому повідомлень. З базових типів можуть бути сконструйовані більш складні типи даних.

Коди завершення

Коди завершення повертаються в якості значення функції або через останній аргумент процедури FORTRAN. Виняток становлять підпрограми MPI_wtime і MPI_wtick, в яких повернення коду помилки не передбачено.

Використовуються стандартні значення MPI_SUCCESS — при успішному завершенні виклику і MPI_ERR_OTHER — зазвичай при спробі повторного виклику процедури MPI_ INIT.

Системою підтримуються і інші помилки. Замість числових кодів в програмах зазвичай використовують спеціальні іменовані константи. Серед них:

- MPI_ERR_BUFFER — неправильний вказівник на буфер;
- MPI_ERR_COMM — неправильний комунікатор;
- MPI_ERR_RANK — неправильний ранг;
- MPI_ERR_OP — неправильна операція;
- MPI__ERR_ARG — неправильний аргумент;
- MPI_ERR_UNKNOWN — невідома помилка;
- MPI_ERR_TRUNCATE — повідомлення обрізано при прийомі;
- MPI_ERR_INTERN — внутрішня помилка. Зазвичай виникає, якщо системі не вистачає пам'яті.

Як влаштована MPI-програма

На початку програми, відразу після заголовка, необхідно підключити відповідний заголовковий файл. В програмі на мові С це mpi.h:

#include "mpi.h"

У цьому файлі містяться описи констант і змінних бібліотеки MPI.

Першим викликом бібліотечної процедури MPI в програмі повинен бути виклик підпрограми ініціалізації MPI_ INIT, перед ним може розташовуватися тільки виклик MPI_Initialized, з допомогою якого визначають ініціалізована система MPI чи ні. Виклик процедури ініціалізації виконується тільки один раз.

Процедура ініціалізації створює комунікатор зі стандартним ім'ям MPI_COMM_WORLD. Це ім'я вказується в усіх наступних викликах процедур MPI.

Після виконання всіх обмінів повідомленнями в програмі повинен розташовуватися виклик процедури MPI_FINALIZE(ierr). В результаті цього виклику видаляються структури даних MPI і виконуються інші необхідні дії. Програміст повинен подбати про те, щоб до моменту виклику процедури MPI_ FINALIZE були завершені всі пересилання даних. Після виконання даного виклику інші виклики процедур MPI, включаючи MPI_ INIT, неприпустимі.

Лістинг 6.1. Найпростіша MPI-програма на мові С

```
#include "mpi.h"

#include <stdio.h>

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

{

int myid, numprocs;

MPI_INIT (&argc, &argv) ;

MPI_COMM_SIZE(MPI_COMM_WORLD, &numprocs);

MPI_COMM_RANK(MPI_COMM_WORLD, &myid) ;

fprintf (stdout, "Process %d of %d\n", myid, numprocs)

MPI_FINALIZE() ;
```

```
return 0;

}
```