Лекція 7. Операції обміну повідомленнями в MPI.

План лекції:

1. Двоточковий обмін повідомленнями
2. Блокуючі операції обміну

    2.1. Стандартний обмін

    2.2. Синхронний блокуючий обмін

    2.3. Буферизований обмін

    2.4. Обмін “по готовності”

3. Підпрограми-пробники
4. Спільні прийом і передача
5. Не блокуючі операції обміну

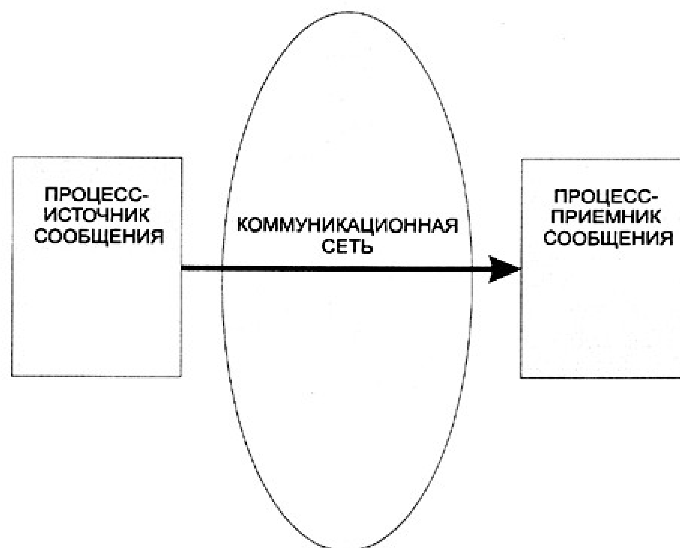
    5.1. Ініціалізація не блокуючого обміну

5.2. Перевірка виконання обміну

6. Відстрочені обміни
7. Скасування "чекаючих" обмінів.

1. Двоточковий обмін повідомленнями

Двоточковий обмін, з точки зору програміста, виконується наступним чином (мал. 7.1): для пересилання повідомлення процес-джерело викликає підпрограму передачі, при зверненні до якої зазначається ранг процесу-одержувача (адресата) у відповідній області взаємодії. Остання визначається своїм комунікатором, зазвичай це MPI_COMM_WORLD. Процес-одержувач, для того щоб отримати спрямоване йому повідомлення, повинен викликати підпрограму прийому, вказавши при цьому ранг джерела.



Мал. 7.1. Двоточковий обмін повідомленнями

У всіх реалізаціях MPI, в тому числі і в MPICH, гарантується виконання деяких властивостей двоточкового обміну:

Одним з них є збереження порядку повідомлень, які при двоточковому обміні не можуть "обганяти" один одного. Якщо, наприклад, процес з рангом 0 передає процесу з рангом 1 два повідомлення: А і В, процес 1 отримає спочатку повідомлення А, а потім В.

Інша найважливіша властивість — гарантоване виконання обміну. Якщо один процес посилає повідомлення, а інший -запит на його прийом, то або передача або прийом будуть вважатися виконаними. При цьому можливі три сценарії обміну:

- другий процес одержує від першого адресоване йому повідомлення;
- надіслане повідомлення може бути отримано третім процесом, при цьому фактично виконана буде передача повідомлення, а не його прийом (повідомлення пройшло повз адресата);
- другий процес одержує повідомлення від третього, тоді передача не може вважатися виконаною, тому що адресат отримав не той "лист".

У двоточковому обміні слід дотримуватися правила відповідності типів переданих і прийнятих даних. Це ускладнює обмін повідомленнями між програмами, написаними на різних мовах програмування.

У MPI є також чотири режими обміну, що розрізняються умовами ініціалізації і завершення передачі повідомлення:

- *стандартна передача* вважається виконаною і завершується, як тільки повідомлення надіслано незалежно від того воно дійшло до адресата чи ні. У стандартному режимі передача повідомлення може починатися, навіть якщо ще не розпочато його прийом;
- *синхронна передача* відрізняється від стандартної тим, що вона не завершується доти, поки не буде завершено прийом повідомлення". Адресат, отримавши повідомлення, посилає процесу, який відправив його, повідомлення, яке має бути отримано відправником для того, щоб обмін вважався виконаним. Операцію передачі повідомлення іноді називають "рукостисканням";
- *буферизована передача* завершується відразу ж, повідомлення копіюється в системний буфер, де і очікує своєї черги на пересилку. Завершується буферизована передача незалежно від того, виконаний прийом повідомлення чи ні;
- *передача "по готовності"* починається тільки в тому випадку, коли адресат ініціалізував прийом повідомлення, а завершується відразу, незалежно від того, прийнято повідомлення чи ні.

Кожен з цих чотирьох режимів є як у блокуючій, так і в не блокуючій формі.

У MPI прийняті наступні угоди про імена підпрограм доточкового обміну:

MPI_[I][R, S, B]Send

тут префікс [I] (Immediate) позначає не блокуючий режим. Один з префіксів [R, S, B] позначає режим обміну, відповідно: по готовності, синхронний і буферизований. Відсутність префікса позначає підпрограму стандартного обміну. Таким чином є 8 різновидів операції передачі повідомлень.

Для підпрограм прийому:

MPI_[I]Recv, тобто всього 2 різновиди прийому.

Підпрограма MPI_Irsend, наприклад, виконує передачу "по готовності" в не блокуючому режимі, MPI_Bsend -буферизовану передачу з блокуванням, а MPI_Recv виконує блокуючий прийом повідомлень.

Підпрограма прийому будь-якого типу може прийняти повідомлення від будь-якої підпрограми передачі.

2. Блокуючі операції обміну

Блокуючі операції призупиняють виконання вхідного процесу, змушуючи процес очікувати завершення передачі даних. Блокування гарантує виконання дій у заданому порядку, але і створює умови для виникнення тупикових ситуацій, коли обидва процеси-учасника обміну "точка-точка" блокуються одночасно.

2.1. Стандартний обмін

Повідомлення, відправлене в стандартному режимі, може протягом деякого часу "гуляти" по комунікаційній мережі паралельного комп'ютера, збільшуючи тим самим її завантаження. У MPI-програми рекомендується дотримувати наступних правил:

- блокуючі операції обміну слід використовувати обережно, оскільки в цьому випадку зростає ймовірність тупикових ситуацій;
- якщо для правильної роботи процесу має значення послідовність прийому повідомлень, джерело повинно передавати нове повідомлення, тільки переконавшись, що попереднє вже прийнято. В іншому випадку може порушитися детермінізм виконання програми;
- повідомлення повинні гарантовано і з досить великою частотою прийматися процесами, яким вони надсилаються, інакше комунікаційна мережа може переповнитися, що призведе до різкого падіння швидкості її роботи. Це, в свою чергу, знизить продуктивність всієї системи.

Основними підпрограма стандартного двоточкового обміну повідомленнями є MPI_Send і MPI_Recv. MPI_send — це підпрограма стандартної блокуючої передачі. Вона блокує виконання процесу до завершення передачі повідомлення (що, однак, не гарантує завершення прийому):

Стандартна блокуюча передача виконується підпрограмою:

int MPI_Send(void *buf, int count, MPI_Datatype datatype, int dest,

int tag, MPI_Comm comm)

Вхідні параметри підпрограми MPI_Send:

- buf — адреса першого елемента у буфері передачі;
- count — кількість елементів у буфері передачі;
- datatype — тип MPI кожного елемента, що пересилається;
- dest — ранг процесу-одержувача повідомлення. Ранг тут — ціле число від 0 до п — 1, де п — число процесів в області взаємодії;
- tag — тег повідомлення;
- сомм — комунікатор;
- ierr — код завершення.

За замовчуванням реакцією системи на виникнення помилки під час виконання програми є її зупинка, однак реакцію можна змінити за допомогою виклику підпрограми MPI_Errhandler_set. Слід мати на увазі, що MPI не гарантує нормального продовження роботи програми після виникнення помилки.

При виклику підпрограм обміну можуть бути такі помилки:

- MPI_ERR_COMM — неправильно вказано комунікатор. Часто виникає при використанні "порожнього" комунікатора;
- MPI_ERR_COUNT — неправильне значення аргументу count (кількість значень, що пересилаються);
- MPI_ERR_TYPE — неправильне значення аргументу, що задає тип даних;
- MPI_ERR_TAG — неправильно вказано тег повідомлення;
- MPI_ERR_RANK — неправильно вказано ранг джерела або одержувача повідомлення;
- MPI_ERR_ARG — неправильний аргумент, помилкове завдання якого не потрапляє ні в один клас помилок;
- MPI_ERR_REQUEST — неправильний запит на виконання операції.

Стандартний блокуючий прийом виконується підпрограмою:

int MPI_Recv(void *buf, int count, MPI_Datatype datatype, int source,

int tag, MPI_Comm comm, MPI_Status *status)

Її вхідні параметри:

- count -- максимальна кількість елементів у буфері прийому. Фактичну їх кількість можна визначити за допомогою підпрограми MPI_Get_count;
- datatype — тип прийнятих даних. Обов'язкове дотримання відповідності типів аргументів підпрограм прийому і передачі;

- source — ранг джерела. Можна використовувати спеціальне значення MPI_ANY_SOURCE, що відповідає довільному значенню рангу. У програмуванні ідентифікатор, що відповідає довільному значенню параметра, часто називають "джокером";
- tag — тег повідомлення або "джокер" MPI_ANY_TAG, що відповідає довільному значенню тегу;
- comm — комунікатор. При вказівці комунікатора "джокери" використовувати не можна.

Слід мати на увазі, що при використанні значень MPI_ANY_SOURCE (будь-яке джерело) і MPI_ANY_TAG (будь-який тег) є небезпека прийому повідомлення, не призначеного даному процесу.

Вихідними параметрами є:

- buf — початкова адреса буфера прийому. Його розмір повинен бути достатнім, щоб розмістити прийняте повідомлення, інакше при виконанні прийому відбудеться збій — виникне помилка переповнення;
- status — статус обміну.

Якщо повідомлення менше, ніж буфер прийому, змінюється вміст лише тих осередків пам'яті буфера, які відносяться до повідомлення. Інформація про довжину отриманого повідомлення міститься в одному з полів статусу, але до цієї інформації у програміста немає прямого доступу (як до поля структури або елементу масиву).

Розмір отриманого повідомлення (count) можна визначити з допомогою виклику підпрограми MPI_Get_count:

```
int MPI_Get_count(MPI_Status *status, MPI_Datatype datatype, int *count)
```

Аргумент datatype повинен відповідати типу даних, зазначеному в операції обміну.

Підпрограма MPI_Recv може приймати повідомлення, відправлені в будь-якому режимі. Є деяка асиметрія між операціями прийому та передачі. Вона полягає в тому, що прийом може виконуватися від довільного процесу, а в операції передачі повинна бути вказана цілком певна адреса. Приймач може використовувати "джокери" для джерела і для тегу. Процес може відправити повідомлення і самому собі, але слід враховувати, що в цьому випадку блокуючі операції можуть призвести до "глухого кута".

Перед завершенням програми та виконанням MPI_Finalize слід переконатися, що прийняті всі повідомлення, надіслані раніше.

Лістинг 7.1. Приклад 1 використання стандартного блокуючого обміну

```
#include "mpi.h"
#include <stdio.h>
```

```
int main(int argc, char *argv[])
{
int myid, numprocs; char message[24] ;
int myrank; MPI_Status status;
int TAG = 0;
MPI_Init(&argc, &argv);
MPI_Comm_rank (MPI_COMM_WORLD, &myrank) ;
if (myrank == 0) {
strcpy(message, "Hi, Parallel Programmer!");
MPI_Send(&message, 25, MPI_CHAR, 1, TAG, MPI_COMM_WORLD); }
else {
MPI_Recv(&message, 25, MPI_CHAR, 0,
TAG, MPI_COMM_WORLD, &status);
printf("received: %s\n", message); }
MPI_Finalize();
return 0;
}
```

У даному прикладі процес з рангом 0 передає повідомлення процесу з рангом 1, використовуючи для цього підпрограму MPI_Send. При виконанні цієї операції в пам'яті процесу-відправника виділяється буфер передачі, який містить змінну message. У буфері містяться 24 символи. Процес з рангом 1 приймає повідомлення з допомогою підпрограми MPI_Recv. Повідомлення зберігається в буфері прийому. Перші три аргументи підпрограми прийому визначають положення, розмір і тип буфера прийому.

Результат виконання цієї програми:

```
received: Hi, Parallel Programmer!
```

Приклад програми, в якому обмінюються повідомленнями процеси з парними і непарними рангами, дано в лістингу 4.2. Передбачається, що значення size парно.

Лістинг 7.2. Приклад 2 використання стандартного блокуючого обміну

```
#include "mpi.h"
#include <stdio.h>
int main(int argc, char *argv[])
{
int myrank, size, message;
int TAG = 0;
```

```
MPI_Status status;
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
MPI_Comm_size(MPI_COMM_WORLD, &size);
message = myrank;
if((myrank % 2) == 0)
{
if((myrank + 1) != size) MPI_Send(&message, 1, MPI_INT, myrank + 1, TAG,
MPI_COMM_WORLD);
}
else
{
if(myrank != 0)
MPI_Recv(&message, -1, MPI_INT, myrank-1, TAG, MPI_COMM_WORLD,
&status);
printf("received :%i\n", message);
}
MPI_Finalize();
return 0;
}
```

Результат виконання цієї програми у разі запуску процесів 10 виглядає так:

received :0 received :2 received :6 received :4 received :8

Порядок виведення повідомлень може бути іншим, він змінюється від випадку до випадку.

Якщо стандартна передача не може бути виконана з-за недостатнього обсягу приймального буфера, здійснення процесу блокується до тих пір, поки не буде доступний буфер достатнього розміру. Іноді це може виявитися зручним. Наведемо приклад. Нехай джерело циклічно посилає нові значення адресату і нехай вони виробляються швидше, ніж споживач може їх прийняти. При використанні буферизованої передачі може виникнути переповнення буфера прийому. Для того щоб його уникнути, в програму доведеться включити додаткову синхронізацію, а при використанні стандартної передачі така синхронізація виконується автоматично. Іноді недостатній розмір буфера може призвести до тупикової ситуації.

2.2. Синхронний блокуючий обмін

При синхронному обміні адресат посилає джерелу "квитанцію" - повідомлення про завершення прийому. Тільки після отримання цього повідомлення обмін вважається завершеним і джерело "знає", що його повідомлення отримано.

Синхронна передача виконується з допомогою підпрограми MPI_Ssend:

int MPI_Ssend(void *buf, int count, MPI_Datatype datatype, int dest,

int tag, MPI_Comm comm)

Її параметри збігаються з параметрами підпрограми MPI_send.

Якщо процес виконує блокуючу синхронну передачу до того, як інший процес спробує отримати повідомлення, він зупиняється до прийому повідомлення адресатом. Синхронний обмін повільніше, ніж стандартний, але він безпечніше, оскільки не дає комунікаційної мережі переповнитися "втраченими в мережі" повідомленнями, які не дійшли до адресата. Цим забезпечується більша ступінь передбачуваності поведінки паралельної програми. Завдяки відсутності "невидимих" повідомлень, простіше виявляється і налагодження програми.

Передачу повідомлення в синхронному режимі може бути розпочато незалежно від того був зареєстрований його прийом чи ні, але вона вважається успішно виконаною тільки при наявності відповідного прийому. Завершення синхронної передачі означає не тільки те, що буфер передачі можна використовувати заново, але і те, що адресат розпочав виконання прийому. Якщо і передача і прийом є блокуючими операціями, обмін не завершиться, поки обидва процеси не почнуть передачу і прийом повідомлень. Операція передачі в цьому режимі не локальна.

2.3. Буферизований обмін

Передача повідомлення в буферизованому режимі може бути розпочата незалежно від того, чи зареєстрований відповідний прийом. Джерело копіює повідомлення в буфер, а потім передає його у не блокуючому режимі так само, як в стандартному режимі. Ця операція локальна, оскільки її виконання не залежить від наявності відповідного прийому. Якщо обсяг буфера недостатній, виникає помилка. Виділення буфера і його розмір контролюються програмістом.

Буферизована передача завершується відразу, оскільки повідомлення негайно копіюється в буфер для подальшої передачі. На відміну від стандартного обміну, в цьому випадку робота джерела і адресата не синхронізується.

Параметри підпрограми буферизованого обміну MPI_Bsend:

int MPI_Bsend(void *buf, int count, MPI_Datatype datatype, int dest,

int tag, MPI_Comm comm)

такі ж, як і у MPI_Send.

При виконанні буферизованого обміну програміст повинен заздалегідь створити буфер достатнього розміру. Це робиться за допомогою виклику підпрограми:

int MPI_Buffer_attach(void *buf, size)

У результаті виклику створюється буфер buf розміром size байтів.

За один раз до процесу може бути підключений тільки один буфер.

Після завершення роботи з буфером його необхідно відключити. Робиться це за допомогою виклику підпрограми:

int MPI_Buffer_detach(void *buf, int *size)

Ця операція блокує роботу процесу до тих пір, поки всі повідомлення, що знаходяться в буфері, не будуть оброблені. Завдяки цьому, виклик даної підпрограми можна використовувати для форсованої передачі повідомлень. Після завершення виклику можна знову використовувати пам'ять, яку займав буфер, проте слід пам'ятати, що в мові С цей виклик не звільняє автоматично пам'ять, відведену для буфера. Приклад використання підпрограм підключення і відключення буфера приведений в лістингу 7.3.

Лістинг 7.3. Приклад використання буферизованого обміну

```
#include "mpi.h"
#include <stdio.h>
int main(int argc,char *argv[])
{
int *buffer;
int myrank;
MPI_Status status;
int buffsize = 1;
int TAG = 0;
MPI_Init(&argc, Sargv);
MPI_Comm_rank(MPI_COMM_WORLD, anyrank);
if (myrank == 0}
{
buffer = (int *) malloc(buffsize + MPI_BSEND_OVERHEAD);
MPI_Buffer_attach(buffer, buffsize +MPI_BSEND_OVERHEAD);
buffer = (int *) 10;
MPI_Bsend(&buffer, buffsize, MPI_INT, 1, TAG,
MPI_COMM_WORLD); MPI_Buffer_detach(&buffer, Sbuffsize);
```

```
} else
{
MPI_Recv(&buffer, buffsize, MPI_INT, 0,
TAG, MPI_COm_WORLD, Sstatus) ;
printf("received: %i\n", buffer);
}
MPI_Finalize() ;
return 0;
}
```

Розмір буфера повинен перевершувати розмір повідомлення на величину MPI_BSEND_OVERHEAD. Цей додатковий простір використовується підпрограмою буферизованої передачі для своїх цілей.

Якщо перед виконанням операції буферизованого обміну не виділено буфер, MPI веде себе так, як якщо б був пов'язаний з процесом буферу нульового розміру. Робота з таким буфером зазвичай завершується збоєм програми.

Буферизований обмін рекомендується використовувати в тих ситуаціях, коли програмісту потрібно мати більший контроль над розподілом пам'яті. Цей режим зручний і для налагодження, оскільки причину переповнення буфера визначити легше, ніж причину глухого кута.

2.4. Обмін "по готовності"

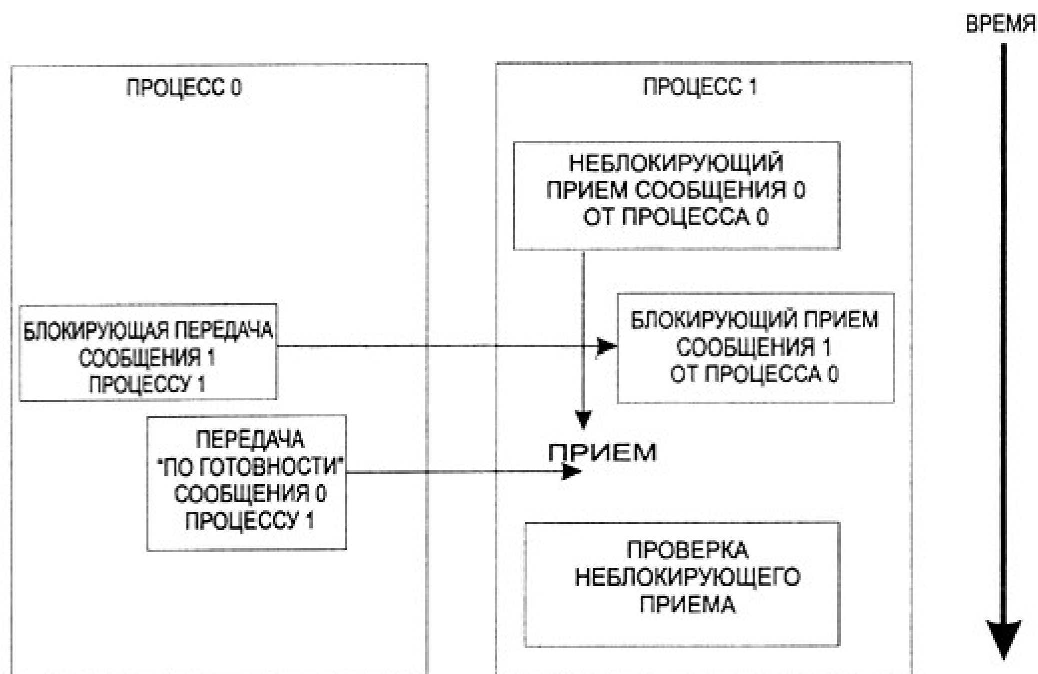
Передача "по готовності" повинна починатися, якщо вже зареєстрований відповідний прийом. Завершується вона відразу ж. Якщо прийом не зареєстрований, результат виконання операції не визначений. Завершення передачі не залежить від того, викликана іншим процесом підпрограма прийому даного повідомлення чи ні, воно означає лише, що буфер передачі можна використовувати знову. Повідомлення просто викидається в комунікаційну мережу в надії, що адресат його почує. Слід мати на увазі, що ця надія може і не збутися. "Правила гри" тут такі ж, як і в операціях стандартного або синхронного обміну. У "правильній" програмі передача "по готовності" може бути замінена стандартної передачею, результат виконання і поведінка програми при цьому не повинні змінитися. Змінитися може лише швидкість її виконання.

Передача "по готовності" виконується з допомогою підпрограми MPI_Rsend:

```
int MPI_Rsend(void *buf, int count, MPI_Datatype datatype, int dest,
int tag, MPI_Comm comm)
```

Параметри у неї ті ж, що і у підпрограми MPI_send.

Обмін "по готовності" може збільшити продуктивність програми, оскільки тут не використовуються етапи встановлення міжпроцесних зв'язків, а також буферизація. Всі ці операції вимагають часу. З іншого боку, обмін "по готовності" потенційно небезпечний, крім того, він ускладнює налагодження, тому його рекомендується використовувати тільки в тому випадку, коли правильна робота програми гарантується її логічною структурою (мал. 7.2), а виграша у швидкодії треба домогтися будь-якою ціною.



Мал. 7.2. "Безпечна" структура програми з використанням обміну "по готовності"

3. Підпрограми-пробники

Отримати інформацію про повідомлення ще до його приміщення в буфер прийому можна за допомогою підпрограм-пробників (або зондувальних підпрограм) MPI_Probe і MpI_Iprobe. На підставі отриманої інформації приймається рішення про подальші дії. Можна, наприклад, виділити буфер прийому достатнього розміру, визначивши довжину повідомлення за допомогою підпрограми MPI_Get_count, яка дає доступ до одного з полів статусу. Якщо після виклику MPI_Probe слідує виклик MPI_Recv з такими ж значеннями аргументів, що і MPI_probe, він помістить у буфер прийому те ж саме повідомлення, інформація про який була отримана MPI_Probe.

Підпрограма MPI_probe виконує блокуючу перевірку доставки повідомлення:

```
int MPI_Probe(int source, int tag, MPI_Comm comm, MPI_Status *status)
```

Її вхідні параметри:

- source — ранг джерела або "джокер";
- tag — значення тегу або "джокер";
- comm — комунікатор.

Вихідним параметром є статус (status), який містить необхідну інформацію.

Не блокуюча перевірка повідомлення виконується підпрограмою MPI_Iprobe:

int MPI Iprobe(int source, int tag, MPI_Comm comm, int *flag,MPI_Status *status)

Вхідні параметри ті ж, що і у підпрограми MPI probe, а вихідні: flag — прапор і status — статус. Якщо повідомлення вже надійшло і може бути прийнято, повертається значення прапора "істина". Перевірка прийому може виконуватися з допомогою MPI_Iprobe неодноразово. Не обов'язково виконувати прийом повідомлення відразу ж після його надходження.

У прикладі з лістингу 7.4 підпрограми-пробники приймають повідомлення від невідомого джерела, які до того ж містять невідому кількість елементів цілого типу. У цьому випадку замість рангу джерела та тегу повідомлення використовуються "джокери". Спочатку за допомогою виклику підпрограми MPI_Probe фіксується надходження (але не прийом!) повідомлення. Потім визначається джерело повідомлення, з допомогою виклику MPi_Get_count визначається його довжина, виділяється буфер відповідного розміру і виконується прийом повідомлення.

Лістинг 7.4. Приклад використання зондувальних підпрограм

```
#include "mpi.h"
#include <stdio.h>
int main(int argc,char *argv[])
{
int myid, numprocs, **buf, source, i;
int message[3] = {0, 1, 2};
int myrank, data = 2002, count, TAG = 0;
MPI_Status status;
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
if (myrank == 0)
{
MPI_Send(&data, 1, MPI_INT, 2, TAG, MPI_COMM_WORLD);
}
else if (myrank == 1)
{
MPI_Send(&message, 3, MPI_INT, 2, TAG, MPI_COMM_WORLD);
}
```

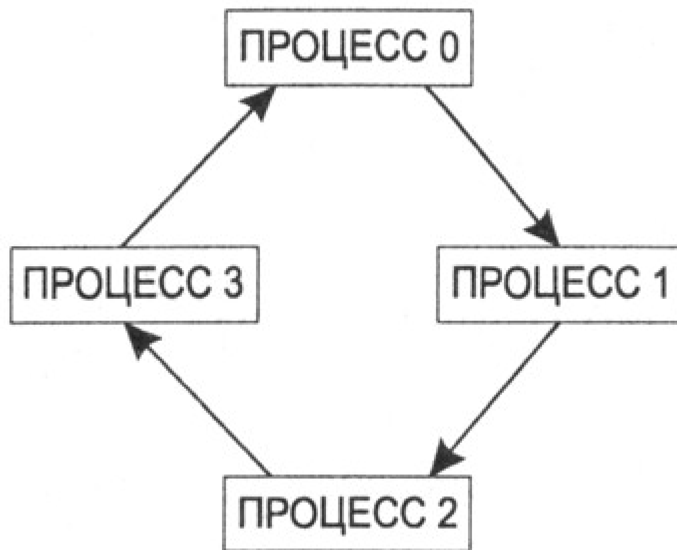
```
else {
MPI_Probe(MPI_ANY_SOURCE, 0, MPI_COMM_WORLD, &status);
source = status.MPI_SOURCE;
MPI_Get_count(&status, MPI_INT, &count);
for (i = 0; i < count; i++){ buf[i] = (int *)malloc(count*sizeof(int));
}
MPI_Recv(&buf[0], count, MPI_INT, source, TAG, MPI_COMM_WORLD, Sstatus);
for (i = 0; i < count; i++){
printf ("received: %d\n", buf[i]);
MPI_Finalize() ;
 return 0;
}
```

При запуску цієї програми повинні створюватися три процеси:

Процес з рангом 0 передає процесу 2 повідомлення, яке містить одне значення (змінна data), а процес з рангом 1 передає тому ж процесу три елемента даних у масиві message. Порядок надходження цих повідомлень не визначений і першим може прийти як повідомлення від 0 процесу, так і повідомлення від процесу 1. Для того щоб правильно організувати прийом повідомлення, необхідно заздалегідь дізнатися його довжину і можливо джерело. Це і роблять підпрограми MPI_Probe і MPI_Get_count. Зауважимо, що в даному прикладі ми свідомо порушили раніше дані рекомендації, — адже з двох надісланих повідомлень буде прийнято лише одне, а таких ситуацій слід уникати.

4. Спільні прийом і передача

Операції прийомо-передачі об'єднують в єдиному виклику передачу повідомлення одному процесу і прийом повідомлення від іншого процесу. Даний вид обмінів може виявитися корисним при виконанні складних схем обміну повідомленнями, наприклад, у ланцюгу процесів (рис. 4.5). Якщо для зсуву застосовуються блокуючі операції передачі і прийому, слід визначити їх правильний порядок, інакше взаємні залежності між процесами можуть призвести до тупикових ситуацій. Парні процеси, наприклад, можуть працювати спочатку на передачу, а непарні на прийом. Потім вони міняються ролями. При використанні операцій прийомо-передачі сама система піклується про дотримання правильної послідовності обмінів.



Мал 7.3. Обмін повідомленнями по ланцюгу процесів, замкнутому в кільце

Підпрограми прийомо-передачі можуть взаємодіяти зі звичайними підпрограмами обміну і підпрограмами зондування.

Підпрограма MPI_sendrecv виконує прийом і передачу даних з блокуванням:

int MPI_Sendrecv(void *sendbuf, int sendcount, MPI_Datatype sendtype,

int dest, int sendtag, void *recvbuf, int recvcount,

MPI_Datatype recvtype, int source, int recvtag, MPI_Coinm comm,

MPI_Status *status)

Її вхідні параметри:

- sendbuf — початкова адреса буфера передачі;
- sendcount — кількість переданих елементів;
- sendtype — тип переданих елементів;
- dest — ранг адресата;
- sendtag — тег переданого повідомлення;
- recvbuf — початкова адреса буфера прийому;
- recvcount — кількість елементів у буфері прийому;
- recvtype — тип елементів у буфері прийому;
- source — ранг джерела;
- recvtag — тег прийнятого повідомлення;
- comm — комунікатор.

Вихідні параметри: recvbuf — початкова адреса буфера прийому і status-операції прийому. І прийом і передача використовують один і той же комунікатор. Буфери передачі і прийому не повинні перетинатися, у них може бути різний розмір, типи прийнятих даних і ті, що пересилаються і також можуть різнитися.

Підпрограма MPI_sendrecv_replace відправляє і приймає повідомлення у блокувальному режимі, використовуючи загальний буфер для передачі і для прийому:

```
int MPI_Sendrecv_replace(void *buf, int count,
MPI_Datatype datatype, int dest, int sendtag,
int source, int recvtag, MPI_Comm comm, MPI_Status *status)
```

Вхідні параметри:

- count — кількість відправлених даних і ємність буфера прийому;
- datatype — тип даних в буфері прийому і передачі;
- dest — ранг адресата;
- sendtag — тег переданого повідомлення;
- source — ранг джерела;
- recvtag — тег прийнятого повідомлення;
- comm — комунікатор.

Вихідні параметри: buf — початкова адреса буфера прийому і передачі та статус (status). Отримане повідомлення не повинно перевищувати за розміром надісланого повідомлення, а дані, які передаються і приймаються повинні бути одного типу. Послідовність прийому і передачі вибирається системою автоматично.

Більшість реалізацій MPI гарантують дотримання певних властивостей двоточкового обміну. Найважливішим з них є збереження порядку повідомлень при прийомі. Ця властивість забезпечує детерміновану поведінку паралельної програми у випадку, коли процеси виконуються в одному потоці, а операції прийому не використовують "джокер" MPI_ANY_SOURCE. Порушити детерміновану поведінку програми можуть виклики підпрограм MPI_Cancel і MPI_Waitany.

MPI не гарантує "справедливої" обробки обмінів. Наведемо приклад. Нехай процесом 0 процесу 1 надіслано повідомлення. Може виявитися, що адресат здійснює періодично повторювані запити на прийом, тим не менш, не виключено, що повідомлення від 0 процесу ніколи не буде прийнято, тому що кожен раз раніше нього буде приходити інше повідомлення, відправлене іншим процесом. Програміст повинен уникати подібних ситуацій.

Слід враховувати і обмеженість ресурсів комп'ютера. Будь-яке повідомлення, що знаходиться в комунікаційній мережі, використовує системні ресурси. Ці ресурси обмежені і при несприятливому збігу обставин їх може виявитися недостатньо. Таких ситуацій слід уникати. Операція буферизованої передачі, яка не може бути виконана із-за недостатнього об'єму буфера, може привести до аварійної зупинки програми.

5. Не блокуючі операції обміну

Для того щоб уникнути простоїв під час виконання обміну, використовують не блокуючі операції. Виклик підпрограми не блокуючої передачі ініціює, але не завершує її. Завершитися виконання підпрограми може ще до того, як повідомлення буде скопійовано в буфер передачі. Застосування не блокуючих операцій покращує продуктивність програми, оскільки в цьому випадку допускається перекриття (тобто одночасне виконання) обчислень та обмінів. Передача даних з буфера або їх зчитування може відбуватися одночасно з виконанням процесом іншої роботи. Для завершення обміну потрібно виклик додаткової процедури, яка перевіряє, чи скопійовані дані в буфер передачі. Незважаючи на те, що при не блокуючому обміні повернення з підпрограми обміну відбувається відразу, запис у буфер або зчитування з нього після цього робити не можна, адже повідомлення може бути ще не відправлено або не отримано і робота з буфером може "зіпсувати" його вміст. Таким чином, не блокуючий обмін виконується в два етапи:

1. Ініціалізація обміну.
2. Перевірка завершення обміну.

Поділ цих кроків робить необхідним маркування кожної операції обміну, що дозволяє цілеспрямовано виконувати перевірки завершення відповідних операцій. Для маркування у не блокуючих операціях використовуються ідентифікатори операцій обміну (requests).

Не блокуюча передача може виконуватися у тих самих чотирьох режимах, що і блокуюча: стандартному, буферизованому, синхронному і "по готовності". Передача в кожному з них може бути розпочата незалежно від того, чи був зареєстрований відповідний прийом. У всіх випадках, операція початку передачі локальна, вона завершується відразу ж і незалежно від стану інших процесів.

5.1. Ініціалізація не блокуючого обміну

Ініціалізація не блокуючих стандартних передач виконується за допомогою підпрограми MPI_Isend:

```
int MPI_Isend(void *buf, int count, MPI_Datatype datatype, int dest,
int tag, MPI_Comm comm, MPI_Request *request)
```

Її вхідні параметри аналогічні аргументів підпрограми MPI_Send, а вихідним є параметр request — ідентифікатор операції.

Підпрограма MPI_Issend ініціалізує не блокуючу синхронну передачу даних:

```
int MPI_Issend(void *buf, int count, MPI_Datatype datatype, int dest,
int tag, MPI_Comm comm, MPI_Request *request)
```

Її параметри збігаються з параметрами підпрограми MPI_Isend.

Не блокуюча буферизована передача повідомлення виконується підпрограмою MPI_Ibsend:

int MPI_Ibsend(void *buf, int count,MPI_Datatype datatype, int dest,

int tag, MPI_Comm comm, MPI_Request *request)

Не блокуюча передача "по готовності" виконується підпрограмою MPI_Irsend:

int MPI_Irsend(void* buf, int count, MPI_Datatype datatype, int dest,

int tag, MPI_Comm comm, MPI_request *request)

Параметри всіх підпрограм не блокуючих передач збігаються.

Підпрограма MPI_Irecv починає не блокуючий прийом:

int MPI_Irecv(void *buf, int count,MPI_Datatype datatype, int source,

int tag, MPI_Comm comm, MPI_Request *request)

Призначення аргументів тут таке ж, як і в попередніх підпрограмах, за винятком того, що зазначається ранг не адресата, а джерела повідомлення (source).

Виклики підпрограм не блокуючого обміну формують запит на виконання операції обміну і пов'язують його з ідентифікатором операції request. Запит ідентифікує різні властивості операції обміну, такі як режим, характеристики буфера обміну, контекст, тег і ранг, використовувані при обміні. Крім того, запит містить інформацію про стан операцій обміну, які чекають обробки, і може бути використаний для отримання інформації про стан обміну або для очікування його завершення.

Виклик підпрограми не блокуючої передачі означає, що система може починати копіювання даних з буфера. Джерело не повинен звертатися до буфера передачі під час виконання не блокуючої передачі. Виклик не блокуючого прийому означає, що система може починати запис даних в буфер прийому. Джерело під час передачі і адресат під час прийому не повинні звертатися до буферу. Не блокуюча передача може бути прийнята підпрограмою блокуючого прийому і навпаки.

Не блокуючий обмін можна використовувати не тільки для збільшення швидкості виконання програми, але і в тих ситуаціях, де блокуючий обмін може призвести до "глухого кута ". Уникнути в цьому випадку "глухого кута" можна, використовуючи підпрограму блокуючого обміну MPI_Sendrecv, але блокуючий обмін уповільнює виконання програми, оскільки його швидкість визначається швидкістю роботи комунікаційної мережі, а ця швидкість, як правило, не дуже велика.

І трохи про терміни. Порожнім називають запит з ідентифікатором MPI_REQUEST_NULL. Відкладений запит на виконання операції обміну називають неактивним, якщо він не пов'язаний ні з яким вихідним повідомленням.

Порожнім називають статус, поле тегу якого приймає значення MPI_ANY_TAG, поле джерела повідомлення - MPI_ANY_SOURCE, а виклики підпрограм MPI_Get_count і MPI_Get_eiements повертають нульове значення аргументу count.

5.2. Перевірка виконання обміну

Перевірка фактичного виконання передачі або прийому в не блокуючому режимі здійснюється за допомогою виклику підпрограм очікування, які блокують роботу процесу до завершення операції, або не блокуючих підпрограм перевірки, які повертають логічне значення "істина", якщо операція виконана.

Підпрограма MPI_Wait блокує роботу процесу до завершення прийому або передачі повідомлення:

int MPI_Wait(MPI_Request *request, MPI_Status *status)

Вхідний параметр request — ідентифікатор операції обміну, а вихідний — статус (status). В аргументі status повертається інформація про виконану операцію. Значення статусу для операції передачі повідомлення можна отримати викликом підпрограми MPI_Test_canceled. Можна викликати MPI_Wait з порожнім або неактивним аргументом request. В цьому випадку операція завершується відразу ж з порожнім статусом.

Успішне виконання підпрограми MPI_Wait після виклику MPI_Ibsend передбачає, що буфер передачі можна використовувати знову, тобто дані, що пересилаються, відправлені або скопійовані в буфер, виділений при виклику підпрограми MPI_Buffer_attach. В цей момент вже не можна скасувати передачу. Якщо не буде зареєстрований відповідний прийом, буфер не можна буде звільнити. В цьому випадку можна застосувати підпрограму MPI_Cancel, яка звільнює пам'ять, виділену підсистемі комунікацій.

Підпрограма MP_Test виконує не блокуючу перевірку завершення прийому або передачі повідомлення:

int MPI_Test(MPI_Request *request, int *flag, MPI_Status *status)

Її вхідним параметром є ідентифікатор операції обміну request. Вихідні параметри: flag - "істина", якщо операція, задана ідентифікатором request, виконана, і status — статус виконаної операції.

Якщо при виклику MPI_Test використовується порожній або неактивний аргумент request, операція повертає значення прапора "істина" і порожній статус. Функції MPI_Wait і MPI_Test можна використовувати для завершення операцій прийому і передачі.

У тому випадку, коли відразу кілька процесів обмінюються повідомленнями, можна використовувати перевірки, які застосовуються одночасно до кількох обмінів. Є три типи таких перевірок:

- перевірка завершення всіх обмінів;
- перевірка завершення будь-якого обміну з декількох;
- перевірка завершення заданого обміну з декількох.

Кожна з цих перевірок, у свою чергу, має два різновиди — "очікування" і "перевірка".

Перевірка завершення всіх обмінів виконується підпрограмою MPI_Waitall, яка блокує виконання процесу до тих пір, поки всі операції обміну, пов'язані з активними запитами в масиві requests, не будуть виконані, і повертає статус цих операцій. Статус обмінів міститься в масиві statuses:

int MPI_Waitall(int count, MPI_Request requests[], MPI_Status statuses[])

Тут count — кількість запитів на обмін (розмір масивів requests і statuses).

В результаті виконання підпрограми MPI_waitall запити, сформовані не блокуючими операціями обміну, анулюються, а відповідним елементам масиву присвоюється значення MPI_REQUEST_NULL. Список може містити порожні або неактивні запити. Для кожного з них встановлюється нульове значення статусу.

В разі невдалого виконання однієї або більше операцій обміну підпрограма MPI_Waitall повертає код помилки MPI_ERR_IN_STATUS і привласнює полю помилки статусу значення коду помилки відповідної операції. Якщо операція виконана успішно, полю присвоюється значення MPI_SUCCESS, а якщо не виконана, але і не було помилки — значення MPI_ERR_PENDING. Останній випадок відповідає наявності запитів на виконання операції обміну, що очікують обробки.

Не блокуюча перевірка виконується за допомогою виклику підпрограми MPI_Testall. Він повертає значення прапор (flag) "істина", якщо всі обміни, пов'язані з активними запитами в масиві requests, виконані. Якщо не завершені всі обміни, прапору присвоюється значення "брехня", а масив statuses не визначено:

int MPI_Testall(int count, MPI_Request requests[], int *flag, MPI_Status statuses[])

Кожному статусу, відповідаючому активному запиту, присвоюється значення відповідного статусу обміну. Якщо запит був сформований операцією не блокуючого обміну, він анулюється, а його елементу масиву присвоюється значення MPI_REQUEST_NULL. Кожному статусу, відповідного порожньому або неактивного запитом, присвоюється нульове значення.

Перевірки завершення будь-якого числа обмінів виконуються з допомогою підпрограм MPI_Waitany і MPI_Testany. Перший з цих варіантів — блокуючий. Виконання

процесу блокується до тих пір, поки, принаймні, один обмін з масиву запитів (requests) не буде завершено. Індекс відповідного елемента в масиві requests повертається в аргументі index, а статус — в аргументі status:

int MPI_Waitany(int count, MPI_Request requests[], int *index, MPI_Status *status)

Вхідні параметри: requests та count — кількість елементів у масиві requests, а вихідні: status і index — індекс запиту (у мові C ціле число від 0 до count – 1)

Якщо запит на виконання операції був сформований не блокуючою операцією обміну, він анулюється і йому присвоюється значення MPI_REQUEST_NULL. Масив запитів може містити порожні або неактивні запити. Якщо в списку взагалі немає активних запитів або він порожній, виклики завершуються відразу зі значенням індексу MPI_UNDEFINED і порожнім статусом.

Підпрограма MPI_Testany перевіряє виконання будь-якого раніше ініціалізованого обміну:

int MPI_Testany(int count, MPI_Request requests[], int *index, int *flag,MPI_Status *status)

Сенс і призначення параметрів цієї підпрограми ті ж, що і для підпрограми MPI_Waitany, але є додатковий аргумент flag, який приймає значення "істина", якщо одна з операцій завершена. Блокуюча підпрограма MPI_Waitany і не блокуюча MPI_Testany взаємозамінні, втім, як і інші аналогічні пари.

Підпрограми MPI_Waitsome і MPI_Testsome діють аналогічно підпрограмам MPI_Waitany і MPI_Testany, крім випадку, коли завершується більше одного обміну. У підпрограмах MPI_waitany і MPI_Testany обмін з числа завершених вибирається довільно, саме для нього і повертається статус, а для MPI_Waitsome і MPI_Testsome статус повертається для всіх завершених обмінів. Ці підпрограми можна використовувати для визначення, скільки обмінів завершено:

int MPI_Waitsome (int incount, MPI_Request requests[], int *outcount, int indices[], MPI_Status statuses[])

Тут incount — кількість запитів. У outcount повертається кількість виконаних запитів з масиву requests, а в перших outcount елементах масиву indices повертаються індекси цих операцій. У перших outcount елементах масиву statuses повертається статус завершених операцій. Якщо виконаний запит був сформований не блокуючою операцією обміну, він анулюється. Якщо в списку немає активних запитів, виконання підпрограми завершується відразу, а параметру outcount присвоюється значення MPI_UNDEFINED.

Підпрограма MPi_Testsome — це не блокуюча перевірка:

int MPI_Testsome (int incount, MPI_Request requests[], int *outcount, int indices[], MPI_Status statuses[])

У неї такі ж параметри, як і у підпрограми MPI_Waitsome. Ефективність підпрограми MPI_Testsome вище, ніж у MPI_Testany, оскільки перша повертає інформацію про всі операції, а для другої потрібен новий виклик для кожної виконаної операції.

6. Відстрочені обміни

Досить часто доводиться стикатися з ситуацією, коли обміни з однаковими параметрами виконуються повторно, наприклад, у циклі. У цьому випадку можна об'єднати аргументи підпрограм обміну в один відкладений запит, який повторно використовується для ініціалізації і виконання обміну повідомленнями. Відкладений запит на виконання не блокуючих операцій обміну дозволяє мінімізувати накладні витрати на організацію зв'язку між процесором і контроллером зв'язку.

Відстрочені запити на обмін об'єднують такі відомості про операції обміну як адреса буфера, кількість пересланих елементів даних, їх тип, ранг адресата, тег повідомлення і комунікатор.

Для створення відкладеного запиту на обмін використовуються чотири підпрограми. Сам обмін вони не виконують. Запит для стандартної передачі створюється при виклику підпрограми MPI_Send_init:

int MPI_Send_init(void *buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm, MPI_Request *request)

Її вхідні параметри:

- buf — адреса буфера передачі;
- count — кількість елементів;
- datatype — тип елементів;
- dest — ранг адресата;
- tag — тег повідомлення;
- comm — комунікатор.

Вихідним параметром є запит на виконання операції обміну (request).

Відкладений запит може бути сформований для всіх режимів обміну.

Для цього використовуються підпрограми MPI_Bsend_init, MPI_Ssend_init і MPI_Rsend_init. Список аргументів у них збігається зі списком аргументів підпрограми MPI_Send_init.

Відкладений обмін ініціалізується подальшим викликом підпрограми MPI_Start:

int MPI_Start(MPI_Request *request)

Вхідним параметром цієї підпрограми є запит на виконання операції обміну (request). Виклик MPI_start із запитом на обмін, створеним MPI_Send_init, ініціює обмін з тими ж властивостями, що і виклик підпрограми MPI_Isend, а виклик MPI_start з запитом, створеним MPI_Bsend_init, ініціює обмін аналогічно викликом MPI_Ibsend. Повідомлення, яке передано операцією, ініційованої з допомогою MPI_start, може бути прийняте будь підпрограмою прийому.

Підпрограма MPI_Startall:

int MPI_Startall(int count, MPI_request *requests)

ініціює всі обміни, пов'язані із запитами на виконання не блокуючіх операцій обміну в масиві requests. Завершується обмін при виклику MPI_wait, MPI_Test та деяких інших підпрограм

7. Скасування "чекаючіх" обмінів

Операція MPI_Cancel дозволяє анулювати не блокуючі "чекаючі" (тобто ті, що очікують обробки) обміни. Її можна використовувати для звільнення ресурсів, насамперед пам'яті, відведеної для розміщення буферів:

int MPI_Cancel(MPI_request *request)

Виклик підпрограми завершується відразу, можливо, ще до реального анулювання обміну. Анульований обмін слід завершити. Зробити це можна за допомогою підпрограм MPI_Request_free, MPI_Wait, MPI_Test і деяких інших.

Операцію MPI_Cancel можна використовувати для анулювання обмінів, які використовують як відкладений, так і звичайний запит. Після виклику MPI_Cancel і наступного за ним виклику MPI_Wait або MPI_Test запит на виконання операції обміну стає неактивним і може бути активізований для нового обміну. Інформація про анульовану операції міститься в аргументі status.

Підпрограма MPI_Test_cancelled повертає значення прапор (flag) "істина", якщо обмін, пов'язаний із зазначеним статусом, успішно анульовано:

int MPI_Test_cancelled(MPI_Status *status, int *flag)

Анулювання — трудомістка операція, не варто ним зловживати. Запит на виконання операції (request) можна анулювати з допомогою підпрограми MPI_Request_free:

int MPI_Request_free(MPI_Request *request)

При виклику ця підпрограма позначає запит на обмін для видалення і присвоює йому значення MPI_REQUEST_NULL. Операції обміну, пов'язаної з цим запитом, дається можливість завершитися, а сам запит видаляється тільки після завершення обміну.

Слід враховувати, що після того, як запит анульовано, не можна перевірити, чи успішно завершено відповідний обмін, тому активний запит не можна анулювати.