Львівський національний університет імені Івана Франка

Факультет електроніки та комп’ютерних технологій

Есе

На тему:

Комунікації між процесами в MPI. Особливості, способи та приклади використання

Виконав

Студент групи ФеП-21

Берніш Микола

Львів 2021

MPI (англ. *Message Passing Interface*, **Інтерфейс передачі повідомлень**) — це специфікація, що була розроблена в 1993–1994 роках групою MPI Forum, і забезпечує реалізацію моделі обміну повідомленнями між процесами У моделі програмування MPI програма породжує кілька процесів, що взаємодіють між собою за допомогою виклику підпрограм прийому й передачі повідомлень.

У MPI існує величезна множина процедур обміну повідомленнями. Вони можуть використовуватися, як для посилки керуючих сигналів, так і для передачі. Два основних види обміну: двухточений і глобальний. Зараз розглянемо двухточковий вид передачі повідомлень.

З погляду програміста, двохточковий обмін виконується в такий спосіб: для пересилання повідомлення процес-джерело викликає підпрограму передачі, при звертанні до якої вказується ранг процесу-одержувача (адресата) у відповідній області взаємодії. Остання задається своїм комунікатором, звичайно це MPI_COMM_WORLD. Процес-одержувач, для того, щоб одержати спрямоване йому повідомлення, викликає підпрограму прийому, указавши при цьому ранг джерела.

Нагадаємо, що MPI гарантує виконання деяких властивостей двохточкового обміну, таких як збереження порядку повідомлень, і гарантоване виконання обміну. Якщо один процес посилає повідомлення, а іншої - 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;

відправлене 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 виконаної, тому що адресат одержав не те "лист".

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

Існують чотири різновиди крапкового обміну: синхронний, асинхронний, блокуючий і неблокуючий. У MPI маються також чотири режими обміну, що розрізняються умовами ініціалізації і завершення передачі повідомлення:

стандартна передача вважається виконаною і завершується, як тільки повідомлення відправлене, незалежно від того, дійшло воно до чи адресата ні. У стандартному режимі передача повідомлення може починатися, навіть якщо ще не початий його прийом;

синхронна передача відрізняється від стандартної тим, що вона не завершується доти, поки не буде довершений прийом повідомлення. Адресат, одержавши повідомлення, посилає процесу, що відправив його, повідомлення, що повинне бути отримане відправником для того, щоб обмін вважався виконаним. Операцію передачі повідомлення іноді називають "рукостисканням";

буферизована передача завершується відразу ж, повідомлення копіюється в системний буфер, де й очікує своєї черги на пересилання. Завершується буферизованна передача незалежно від того, виконаний прийом чи повідомлення ні;

передача "по готовності" починається тільки в тому випадку, коли адресат ініціалізував прийом повідомлення, а завершується відразу, незалежно від того, прийняте чи повідомлення ні.

Кожний з цих чотирьох режимів існує як у що блокуючий, так і в неблокуючій формах. При формі прийому, що блокуючий,/передачі виконання програми припиняється по завершення виконання операції.

У MPI прийняті наступні угоди про імена підпрограм двохточкового обміну: MPI_[I][R|S|B]Send. Префікс I (Immediate) позначає режим, неблокуючий, один із префіксів R|S|B позначає режим обміну по відповідно готовності, синхронний і буферизований, відсутність префікса позначає стандартний обмін. Разом – 8 різновидів передачі повідомлень. Для прийому ж існує всього 2 різновиди: MPI_[I]Recv. Приклади: MPI_Irsend - виконує передачу «по готовності» у режимі, неблокуючий, MPI_Bsend - буферизована передача з блокуванням, MPI_Recv - прийом, що блокуючий. Відзначимо, що підпрограма прийому будь-якого типу може прийняти повідомлення від будь-якої програми передачі. Перейдемо до прикладів:

```c
#include <mpi.h>
#include <stdio.h>
#define TAG_SEND_FWD 99
#define TAG_SEND_BACK 98
#define TAG_REPLY 97

int main(int argc, char* argv[])
{
int k,x;
int myrank, size;
MPI_Status status;
MPI_Init(&argc,&argv);
MPI_Comm_size(MPI_COMM_WORLD,&size);
MPI_Comm_rank(MPI_COMM_WORLD,&myrank);
if (myrank == 0) // призначимо один процес головним
{
puts("Running procs forwards"); fflush(stdout); // негайний вивід повідомлення
x=1;
while (x < size)
{
MPI_Ssend(&x, 1, MPI_INT, x, TAG_SEND_FWD, MPI_COMM_WORLD);
MPI_Recv (&k, 1, MPI_INT, x, TAG_REPLY, MPI_COMM_WORLD, &status);
printf("Reply from proc %d received %dn",x,k); fflush(stdout);
x++;
}
puts("Running procs backwards"); fflush(stdout);
x=size-1;
while (x >0)
{
MPI_Send(&x,1, MPI_INT, x, TAG_SEND_BACK, MPI_COMM_WORLD);
x--;
}
else // інші процеси - підлеглі
{
MPI_Recv(&k, 1, MPI_INT, 0, TAG_SEND_FWD, MPI_COMM_WORLD, &status);
printf("Proc %d received %dn",myrank,k); fflush(stdout);
MPI_Ssend(&k,1, MPI_INT, 0, TAG_REPLY, MPI_COMM_WORLD);
MPI_Recv(&k, 1, MPI_INT, 0, TAG_SEND_BACK, MPI_COMM_WORLD, &status);
printf("Proc %d Received %dn",myrank,k); fflush(stdout);
}
MPI_Finalize();
return 0;
}
```

У цьому прикладі один із процесів (з рангом 0) розсилає повідомлення іншому у прямому, а потім у зворотному порядку.

Прототип функції: int MPI_[..]Send(void* buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm).

Вхідні параметри (однакові для усіх функцій *send):

buf – адреса першого елемента в буфері передачі

count – кількість елементів у буфері передачі

Стандартна передача, що блокуючий, починається незалежно від того, чи був зареєстрований відповідний прийом, а завершується тільки після того, як повідомлення прийняте системою і процес-джерело може знову використовувати буфер передачі.

Повідомлення може бути скопійоване прямо в буфер прийому, а може бути поміщене в тимчасовий системний буфер, де і буде чекати виклику адресатом підпрограми прийому. У цьому випадку говорять про буферизації повідомлення. Передача може завершитися ще до виклику відповідної операції прийому. З іншого боку, буфер може бути недоступний чи MPI може вирішити не буферизувати вихідні повідомлення з міркувань збереження високої продуктивності. У цьому випадку передача завершиться тільки після того, як буде зареєстровані відповідний прийом і дані будуть передані адресату.

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 (у MPI_Irecv – такі ж):

count – максимальна кількість елементів у буфері прийому. Фактична їхня кількість можна визначити за допомогою підпрограми MPI_Get_count;

datatype – тип прийнятих даних. Нагадаємо про необхідність дотримання відповідності типів аргументів підпрограм прийому і передачі;

source – ранг джерела. Можна використовувати спеціальне значення mpi_any_source, що відповідає довільному значенню рангу.

```
Результат виводу(np = 6): Ssend & replies

Running procs forwards

Proc 1 received 1

Reply from proc 1 received 1

Proc 2 received 2

Reply from proc 2 received 2

Proc 3 received 3

Reply from proc 3 received 3

Proc 4 received 4

Reply from proc 4 received 4

Proc 5 received 5

Reply from proc 5 received 5

Running procs backwards

Proc 5 Received 5

Proc 4 Received 4
```

```
Proc 3 Received 3

Proc 2 Received 2

Proc 1 Received 1
```

У двохточковому обміні беруть участь два процеси – джерело повідомлення його адресат. При виконанні колективного обміну кількість "діючих облич" зростає. Повідомлення пересилається від одного процесу декільком чи, навпаки, один процес "збирає" дані від декількох процесів. MPI підтримує такі види колективного обміну, як широкомовну передачу, операції приведення і т.д. У MPI маються підпрограми, що виконують операції розподілу і збору даних, глобальні математичні операції, такі як підсумовування елементів чи масиву обчислення його максимального елемента і т.д.

У будь-якому колективному обміні бере участь кожен процес з деякої області взаємодії. Можна організувати обмін і в підмножині процесів, для цього маються засоби створення нових областей взаємодії і відповідних їм комунікаторів. Колективні обміни характеризуються наступним:

колективні обміни не можуть взаємодіяти з двохточковими. Колективна передача, наприклад, не може бути перехоплена двохточковой підпрограмою прийому;

колективні обміни можуть виконуватися як із синхронізацією, так і без її;

усі колективні обміни є блокирующими для їхній обменаЭЪ, що ініціював;

теги повідомлень призначаються системою.

Широкомовне розсилання

Широкомовне розсилання виконується одним виділеним процесом, що називається головним (root), а всі інші процеси, що приймають участь в обміні, одержують по одній копії повідомлення від головного процесу:

Виконується широкомовне розсилання за допомогою підпрограми

int MPI_Bcast(void *buffer, int count, MPI_Datatype datatype, int root, MPI_Comm comm)

Її параметри одночасно є вхідними і вихідними:

buffer — адреса буфера;

count — кількість елементів даних у повідомленні;

datatype — тип даних MPI;

root — ранг головного процесу, що виконує широкомовне розсилання;

comm — комунікатор.

Обмін із синхронізацією

Синхронізація за допомогою "бар'єра" є найпростішою формою синхронізації колективних обмінів. Вона не вимагає пересилання даних. Підпрограма MPI_Barrier блокуючий виконання кожного процесу з комунікатора comm доти, поки всі процеси не викликають цю підпрограму: int MPI_Barrier(MPI_Comm comm)

Розподіл і збір даних

Розподіл і збір даних виконуються за допомогою підпрограм MPI_Scatter і MPI_Gather відповідно. Список аргументів в обох підпрограм однаковий, але діють вони по-різному.

Повний список підпрограм розподілу і збору даний приведений у таблиці:

Підпрограма	Короткий опис
MPI_Allgather	Збирає дані від усіх процесів і пересилає їх усім процесам
MPI_Allgatherv	Збирає дані від усіх процесів і пересилає їх усім процесам ("векторний" варіант підпрограми MPI_Allgather)
MPI_Allreduce	Збирає дані від усіх процесів, виконує операцію приведення, і результат розподіляє всім процесам
MPI_Alltoall	Пересилає дані від усіх процесів усім процесам
MPI_Alltoallv	Пересилає дані від усіх процесів усім процесам ("векторний" варіант підпрограми MPI_Alltoall)
MPI_Gather	Збирає дані від групи процесів
MPI_Gatherv	Збирає дані від групи процесів ("векторний" варіант підпрограми MPI Gather)
MPI_Reduce	Виконує операцію приведення, тобто обчислення єдинного значення по масиву вихідних даних
MPI_Reduce_scatter	Збір значень з наступним розподілом результата операції приведення
MPI_Scan	Виконання операції сканування (часткова редукція) для даних від групи процесів
MPI_Scatter	Розподіляє дані від одного процесу всім іншим процесам у групі
MPI_Scatterv	Пересилає буфер вроздріб усім процесам у групі ("векторний" варіант підпрограми MPI_Scatter)

Операції приведення і сканування

Операції приведення і сканування відносяться до категорії глобальних обчислень. У глобальній операції приведення до даних від усіх процесів із заданого комунікатора застосовується операція MPI_Reduce

Аргументом операції приведення є масив даних — по одному елементі від кожного процесу. Результат такої операції — єдине значення (тому вона і називається операцією приведення).

У підпрограмах глобальних обчислень функція, передана в підпрограму, може бути: визначеною функцією MPI, наприклад MPI_SUM, користувальницькою функцією, а також оброблювачем для користувальницької функції, що створюється підпрограмою MPI_Op_create.

Три версії операції приведення повертають результат:

одному процесу;

усім процесам;

розподіляють вектор результатів між усіма процесами.

Операція приведення, результат якої передається одному процесу, виконується при виклику підпрограми MPI_Reduce

изначених операцій приведення (див. табл.).

Операція	Опис
MPI_MAX	Визначення максимальних значень елементів одномірних масивів цілого чи речовинного типу
MPI_MIN	Визначення мінімальних значень елементів одномірних масивів цілого чи речовинного типу
MPI_SUM	Обчислення суми елементів одномірних масивів цілого, речовинного чи комплексного типу
MPI_PROD	Обчислення заелементного добутку одномірних масивів цілого, речовинного чи комплексного типу
MPI_LAND	Логічне "И"
MPI_BAND	Бітове "И"
MPI_LOR	Логічне "ЧИ"
MPI_BOR	Бітове "ЧИ"
MPI_LXOR	Логічне "ЧИ", що виключає
MPI_BXOR	Бітове "ЧИ", що виключає
MPI_MAXLOC	Максимальні значення елементів одномірних масивів і їхні індекси
MPI_MINLOC	Мінімальні значення елементів одномірних масивів і їхні індекси

Розглянемо приклад

```
#include <mpi.h>
#include <stdio.h>
#include <math.h>
int main(int argc, char *argv[])
{
int n, myid, numprocs, i;
double PI25DT = 3.14159265358979323846264643;
double mypi, pi, h, sum, x;
MPI_Init(&argc,&argv);
MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
MPI_Comm_rank(MPI_COMM_WORLD,&myid);
while (1) {
if (myid == 0) {
printf("Enter the number of intervals: (0 quits) ");
scanf("%d",&n);
}
```

```
MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD);
if (n == 0)
break;
else {
h = 1.0 / (double) n;
sum = 0.0;
for (i = myid + 1; i <= n; i += numprocs) { //розподіл навантаження
x = h * ((double)i - 0.5);
sum += (4.0 / (1.0 + x*x));
}
mypi = h * sum;
MPI_Reduce(&mypi, &pi, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);
//зборка результату
if (myid == 0)
printf("pi is approximately %.16f, Error is %.16fn",
pi, fabs(pi - PI25DT));
}
MPI_Finalize();
return 0;
}
```

Ця програма обчислює число π методом підсумовування ряду. Спочатку один із процесів (0) запитує число інтервалів, що потім поширює іншим процедурою MPI_Bcast. Помітьте, що процедура MPI_Bcast для процесу 0 є передавальної, а для всіх інших – приймаючої. Кінцеві результати обчислень здаються процесу 0 для підсумовування: процедура

MPI_Reduce(&mypi,&pi,1,MPI_DOUBLE,MPI_SUM,0,MPI_COMM_WORLD)

збирає з усіх процесів перемінну mypi, підсумовує (MPI_SUM), і зберігає результат у змінної pi процесу 0.

Вивід приклада(np = 6):

Process 5 on apc-pc.

Process 3 on apc-pc.

Process 0 on apc-pc.

Enter the number of intervals: (0 quits) Process 1 on apc-pc.

Process 2 on apc-pc.

Process 4 on apc-pc.

15

pi is approximately 3.1419630237914191, Error is 0.0003703702016260

wall clock time = 0.031237

Enter the number of intervals: (0 quits) 2

pi is approximately 3.1623529411764704, Error is 0.0207602875866773

wall clock time = 0.000943

Enter the number of intervals: (0 quits) 0