Львівський національний університет імені Івана Франка

Факультет електроніки та комп’ютерних технологій

Есе

На тему:

Колективні комунікаційні операції в MPI. Приклади застосування операцій колективного обміну даних з інтер- та інтра-комунікаторами

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.

Набір операцій типу точка-точка є достатнім для програмування будь-яких алгоритмів, проте MPI навряд чи б завоював таку популярність, якби обмежувався тільки цим набором комунікаційних операцій.

Однією з найбільш привабливих сторін MPI є наявність широкого набору колективних операцій, які беруть на себе виконання найбільш часто зустрічаються при програмуванні дій. Наприклад, часто виникає потреба розіслати деяку змінну або масив з одного процесора всім іншим.

Кожен програміст може написати таку процедуру з використанням операцій Send / Recv, однак набагато зручніше скористатися колективної операцією MPI_Bcast. Причому гарантовано, що ця операція буде виконуватися набагато ефективніше, оскільки MPI-функція реалізована з використанням внутрішніх можливостей комунікаційного середовища.

Головна відмінність колективних операцій від операцій типу точка-точка полягає в тому, що в них завжди беруть участь всі процеси, пов'язані з деяким комунікатором.

Недотримання цього правила призводить або до аварійного завершення завдання, або до ще більш неприємного зависання завдання.

Набір колективних операцій включає:

- Синхронізацію всіх процесів за допомогою бар'єрів (MPI_Barrier);
- Колективні комунікаційні операції, в число яких входять:
    - розсилка інформації від одного процесу всім іншим членам деякої галузі зв'язку (MPI_Bcast);
    - збірка (gather) розподіленого по процесам масиву в один масив зі збереженням його в адресному просторі виділеного (root) процесу (MPI_Gather, MPI_Gatherv)

    - збірка (gather) розподіленого масиву в один масив з розсилкою його всім процесам певної галузі зв'язку (MPI_Allgather, MPI_Allgatherv)

    - розбиття масиву і розсилка його фрагментів (scatter) всіх процесів галузі зв'язку (MPI_Scatter, MPI_Scatterv);

    - поєднана операція Scatter / Gather (All-to-All), кожен процес ділить дані зі свого буфера передачі і розкидає фрагменти всім іншим процесам, одночасно збираючи фрагменти, послані іншими процесами в свій буфер прийому (MPI_Alltoall, MPI_Alltoallv).

- Глобальні обчислювальні операції (sum, min, max і ін.) Над даними, розташованими в адресних просторах різних процесів:
    - зі збереженням результату в адресному просторі одного процесу (MPI_Reduce);
    - з розсилкою результату всім процесам (MPI_Allreduce);
    - поєднана операція Reduce / Scatter (MPI_Reduce_scatter);
    - префиксная редукція (MPI_Scan);

Всі комунікаційні підпрограми, за винятком MPI_Bcast, представлені в двох варіантах:

- простий варіант, коли всі частини переданого повідомлення мають однакову довжину і займають суміжні області в адресному просторі процесів;
- "Векторний" варіант надає більш широкі можливості по організації колективних комунікацій, знімаючи обмеження, властиві простому варіанту, як в частині довжин блоків, так і в частині розміщення даних в адресному просторі процесів. Векторні варіанти відрізняються додатковим символом "v" в кінці імені функції.

Відмінні риси колективних операцій:

- Колективні комунікації не взаємодіють з комунікаціями типу точка-точка.

- Колективні комунікації виконуються в режимі з блокуванням. Повернення з підпрограми в кожному процесі відбувається тоді, коли його участь у колективній операції завершилося, однак це не означає, що інші процеси завершили операцію.
- Кількість одержуваних даних має дорівнювати кількості надісланих даних.
- Типи елементів посилаються і одержуваних повідомлень повинні збігатися.
- Повідомлення не мають ідентифікаторів.

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

У MPI існує два типи комунікаторів:

intracommunicator - описує область зв'язку деякої групи процесів;
intercommunicator - служить для зв'язку між процесами двох різних груп.

Інтеркоммунікатори і інтракоммунікатори: описатели областей зв'язку відповідно над двома групами або над однією. MPI_COMM_WORLD є інтрАкоммунікатором. Інтер-комунікатори не є предметом першої необхідності для новачка, тому за межами даного абзацу згадок про них немає. Всі згадані в документі функції, які оперують комунікаторами, або не розрізняють "інтра" і "інтер-" зовсім, або явно вимагають "інтра". До числа останніх відносяться:

- ВСІ колективні функції: MPI_Bcast співтовариші;
- функції управління топологиями;
- інформаційні функції типу MPI_Comm_xxx, які реалізуються через MPI_Group_xxx, наприклад, MPI_Comm_size; замість них використовуються MPI_Comm_remote_xxx.

Тип комунікатора можна визначити за допомогою спеціальної функції MPI_Comm_test_inter.

C: MPI_Comm_test_inter (MPI_Comm comm, int * flag)

IN comm - комунікатор;

OUT flag - повертає true, якщо comm - intercommunicator.

Функція повертає значення "істина", якщо комунікатор є inter комунікатором.

При ініціалізації MPI створюється два зумовлених комунікатора:

MPI_COMM_WORLD - описує область зв'язку, що містить всі процеси;
MPI_COMM_SELF - описує область зв'язку, що складається з одного процесу.

Приклад коду з використанням inter та intra комунікаторів:

```
int main(int argc, char ** argv) {
   MPI_Comm myComm; /* intra комунікатор локальної підгрупи */
   MPI_Comm myFirstComm; /* inter коммунікатор */
   MPI_Comm mySecondComm; /* Другий інтер коммунікатор  */
   int membershipKey;
   int rank;
   MPI_Init( & argc, & argv);
   MPI_Comm_rank(MPI_COMM_WORLD, & rank);
   /* Код користувача повинен генерувати membershipKey в діапазоні [0, 1, 2]  */
   membershipKey = rank % 3;
   /* Створення intra коммунікаторів для локальної підгруппи */
   MPI_Comm_split(MPI_COMM_WORLD, membershipKey, rank, & myComm);
   /* Створення inter коммунікаторів */
   if (membershipKey == 0) {
     /* Група 0 коммунікує з групою 1. */
     MPI_Intercomm_create(myComm, 0, MPI_COMM_WORLD, 1,
       1, & myFirstComm);
   } else if (membershipKey == 1) {
     /* Група 1 коммунікує з групами 0 та 2. */
     MPI_Intercomm_create(myComm, 0, MPI_COMM_WORLD, 0,
       1, & myFirstComm);
     MPI_Intercomm_create(myComm, 0, MPI_COMM_WORLD, 2,
       12, & mySecondComm);
   } else if (membershipKey == 2) {
     /* Група 2 коммунікує з групою 1. */
     MPI_Intercomm_create(myComm, 0, MPI_COMM_WORLD, 1,
       12, & myFirstComm);
   }
   /* Виконується робота */
   switch (membershipKey) /* Звільнення коммунікаторів */ {
   case 1:
     MPI_Comm_free( & mySecondComm);
   case 0:
   case 2:
     MPI_Comm_free( & myFirstComm);
     break;
   }
   MPI_Finalize();
 }
```