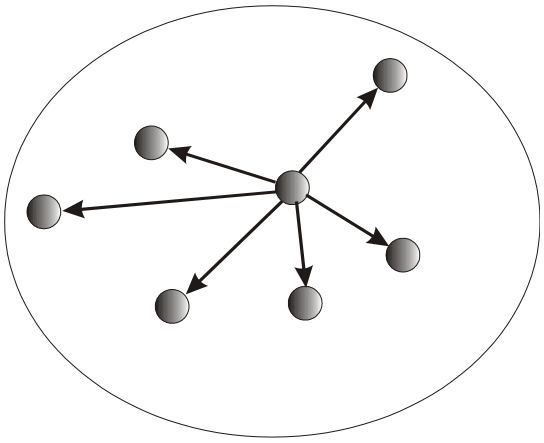
Лекція 8. Колективні операції обміну повідомленнями в MPI.

План лекції:

1. Широкомовна розсилка
2. Обмін з синхронізацією
3. Розподіл і збір даних
4. Векторні підпрограми розподілу даних
5. Операції передачі даних від всіх процесів всім процесам
6. Операції приведення і сканування

У будь-якому колективному обміні бере участь кожен процес з деякої області взаємодії (мал. 8.1). Можна організувати обмін і в підмножині процесів, для цього є засоби створення нових областей взаємодії і відповідних їм комунікаторів.

У MPI є підпрограми, що виконують операції розподілу і збору даних, глобальні математичні операції, такі як додавання елементів масиву або його обчислення максимального елемента і т.п.



Мал. 8.1. Колективний обмін даними

Колективні обміни характеризуються наступними властивостями:

• колективні обміни не можуть взаємодіяти з двоточковими. Колективна передача, наприклад, не може бути перехоплена двоточковою підпрограмою прийому;

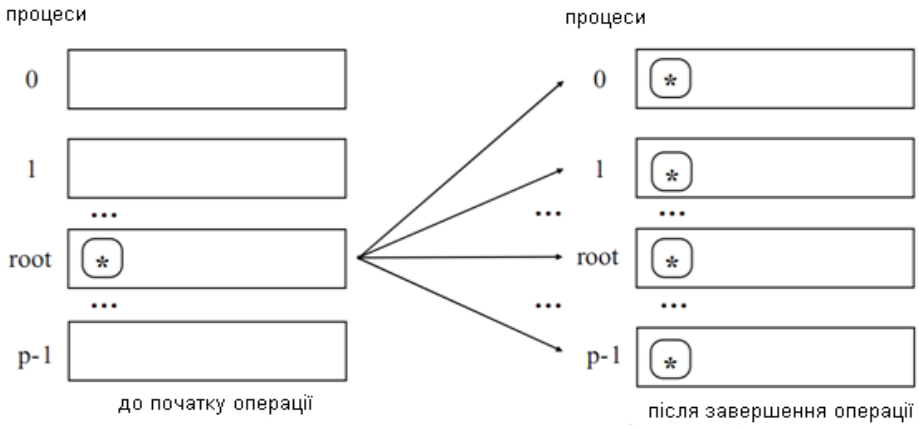
• колективні обміни можуть виконуватися як із синхронізацією, так і без неї;

• всі колективні обміни є блокуючими для обміну, який ініціював їх;

• теги повідомлень призначаються системою.

1. Широкомовна розсилка

Широкомовна розсилка виконується одним виділеним процесом, який називається головним (root), а всі інші процеси, які беруть участь в обміні, отримують по одній копії повідомлення від головного процесу



Мал. 8.2. Широкомовна розсилка

Виконується широкомовна розсилка з допомогою підпрограми MPI_Bcast:

int MPI_Bcast(void *buffer, int count, MPI_Datatype datatype, int root,

MPI_Comm comm)

Її параметри одночасно є вхідними та вихідними:

- buffer — адреса буфера;
- count — кількість елементів даних у повідомленні;
- datatype — тип данихMPI;
- root — ранг головного процесу, що виконує широкомовну розсилку;
- comm — комунікатор.

Особливості:

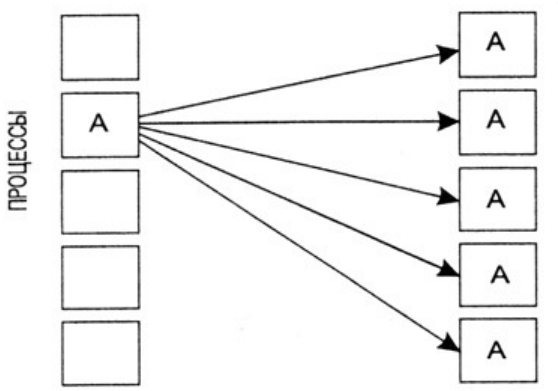
1. Функція MPI_Bcast визначає колективну операцію і, тим самим, при виконанні необхідних розсилок даних виклик функції MPI_Bcast має бути здійснений всіма процесами вказуваного комунікатора.

2. Вказаний у функції Mpi_bcast буфер пам'яті має різне призначення в різних процесах. Для процесу з рангом root, з якого здійснюється розсилка даних, в цьому буфері повинне знаходитися повідомлення, що розсилається. Для всіх останніх процесів вказаний буфер призначений для прийому даних.

2. Обмін з синхронізацією

Синхронізація за допомогою "бар'єру" є найпростішою формою синхронізації колективних обмінів. Вона не вимагає пересилання даних. Підпрограма MPI_Barrier

блокує виконання кожного процессу з комунікатора comm до тих пір, поки всі процеси не викличуть цю підпрограму:



Мал. 8.3. Загальна схема операції сихронізації

Параметром підпрограми є поточний комунікатор:

int MPI_Barrier(MPI_Comm comm.)

Лістинг 8.1. Приклад використання колективного обміну:

```
#include "mpi.h"
#include <stdio.h>
int main(int argc,char *argv[])
{
char data[24];
int myrank, count = 25;
MPI_Status status;
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
if (myrank == 0) {
strcpy(data, "Hi, Parallel Programmer!");
MPI_Bcast (&data, count, MPI_BYTE, 0, MPI_ COMM _WORLD) ;
printf("send: %s\n", data); }
else{
MPI_Bcast(&data, count, MPI_BYTE, 0, MPI_COMM_WORLD);
printf("received: %s\n", data);
}
MPI_Finalize(); return 0;
}
```

Результат виконання цієї програми:

send:Hi, Parallel Programmer!

received: Hi, Parallel Programmer!

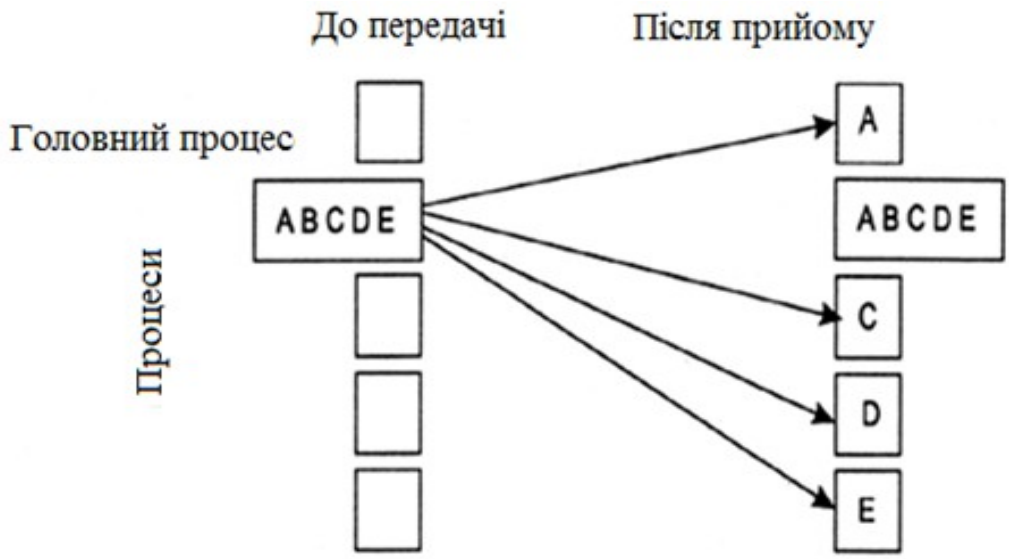
received: Hi, Parallel Programmer!

Лістинг 8.2. Приклад використання широкомовної розсилки

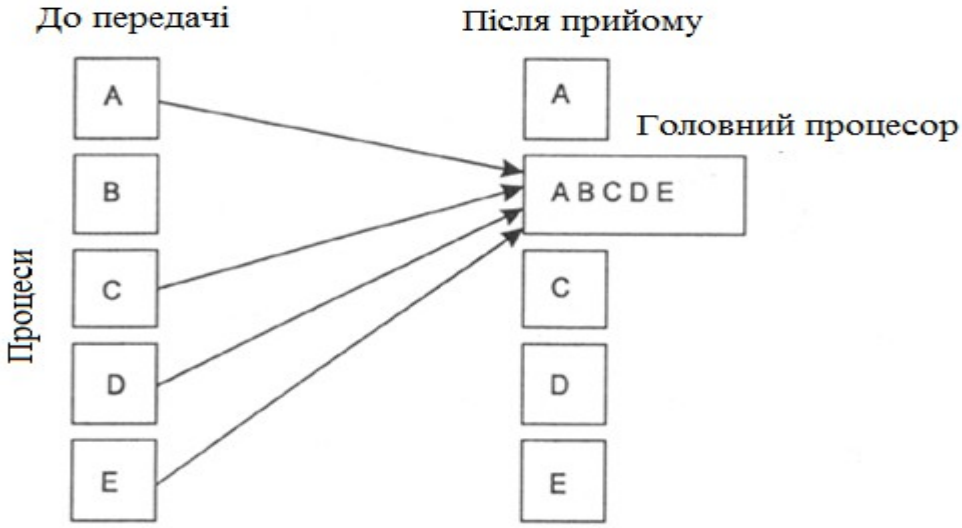
```
#include "mpi.h"
#include<stdio.h>
int main (int argc, char *argv[])
{
int my rank;
int root = 0;
int count = 1;
float a, b;
int n;
MPI_Init (&argc, &argv) ;
MPI_Comm_rank(MPI_COMM_WORLD, &myrank) ;
if (my rank == 0) {
printf ("Enter a, b, n\n"); scanf("%f %f %i", &a, &b, &n) ;
MPI_Bcast (&a, count, MPI_FLOAT, root, MPI_COMM_WORLD)
MPI_Bcast (&b, count, MPI_FLOAT, root, MPI_COMM_WORLD)
MPI_Bcast(&n, count, MPI_INT, root, MPI_COMM_WORLD) ;
}
else
{
MPI_Bcast(&a, count, MPI_FLOAT, root, MPI_COMM_WORLD) ;
MPI Bcast (&b, count, MPI FLOAT, root, MPI COMM_WORLD) ;
MPI_Bcast (&n, count, MPI_INT, root, MPI_COMM_WORLD) ;
printf("%i Process got %f %f %i\n", myrank, a, b, n) }
MPI_Finalize() ;
return 0;
```

3. Розподіл і збір даних

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



Мал. 8.4. Схема передачі даних для операції розподілу даних



Мал. 8.5. Схема передачі даних для операції збору даних

При широкомовній розсилці всім процесам передається один і той же набір даних, а при розподілі передаються його частини.

Виконується розподіл даних підпрограмою MPI_Scatter, яка пересилає дані від одного процесу всім іншим процесам в групі.

int MPI_Scatter(void *sendbuf, int sendcount, MPI_Datatype sendtype, void *rcvbuf, int rcvcount, MPI_Datatype rcvtype, int root, MPI_Comm comm)

Її вхідні параметри:

- sendbuf — адреса буфера передачі;;

- sendcount — кількість елементів, що пересилаються кожному процесу (але не сумарна кількість пересилаються елементів);
- sendtype — тип переданих даних;
- rcvcount — кількість елементів у буфері прийому;
- rcvtype — тип прийнятих даних;
- root — ранг передавального процесу;
- comm — комунікатор.

Вихідний параметр rcvbuf - адреса буфера прийому.

Працює ця підпрограма наступним чином. Процес з рангом root ("головний процес") розподіляє вміст буфера передачі sendbuf серед всіх процесів. Вміст буфера передачі розбивається на кілька фрагментів, кожен з яких містить sendcount елементів. Перший фрагмент передається процесу 0, другий процесу 1 і т.п. Аргументи send мають значення тільки на стороні процесу root.

Особливості:

1.Оскільки функція MPI_Scatter визначає колективну операцію, виклик цієї функції при виконанні розсилки даних має бути забезпечений в кожному процесі коммунікатора.

2.Функція MPI_Scatter передає всім процесам повідомлення однакового розміру. Виконання загальнішого варіанту операції розподілу даних, коли розміри повідомлень для процесів можуть бути різного розміру, забезпечується за допомогою функції MPI_Scatterv.

Підпрограма MPI_Gather виконує збір повідомлень від інших процесів у буфер головного завдання.

int MPI_Gather (void * sendbuf, int sendcount, MPI_Datatype sendtype, void * rcvbuf, int rcvcount, MPI_Datatype rcvtype, int root, MPI_Comm comm)

Кожен процесс в комунікаторі comm пересилає вміст буфера передачі sendbuf процессу з рангом root. Процес root "склеює" отримані дані в буфері прийому. Порядок склеювання визначається рангами процесів, тобто в результуючому наборі після даних від процессу 0 йдуть дані від процессу 1, потім дані від процессу 2 і т.п.

Аргументи rcvbuf, rcvcountircvtype відіграють роль тільки на стороні головного процесу. Аргумент rcvcount вказує кількість елементів даних, отриманих від кожного процесу (але не сумарну їх кількість).

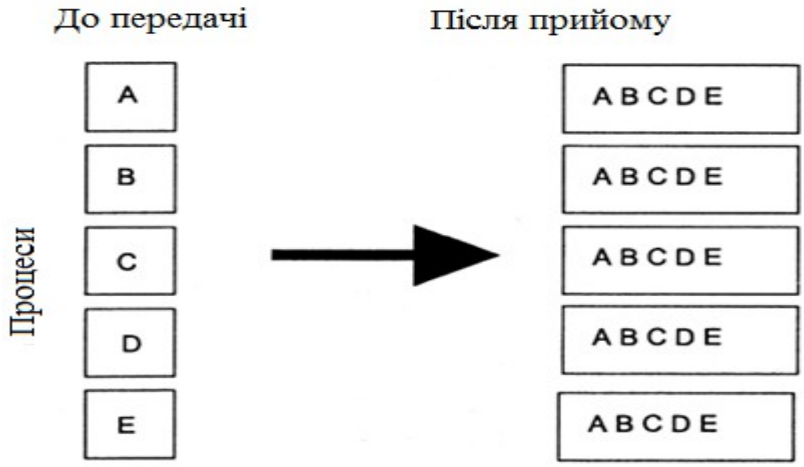
При виклику підпрограм MPI_Scatter і MPI_Gather з різних процесів варто використовувати загальний головний процес.

Особливості:

1. Функція MPI_Gather також визначає колективну операцію, і її виклик при виконанні збору даних має бути забезпечений в кожному процесі коммунікатора.

2. При використанні функції MPI_Gather збірка даних здійснюється лише на одному процесі. Для здобуття всіх збираних даних на кожному з процесів коммунікатора необхідно використовувати функцію збору і розсилки:

Int MPI_Allgather(void *sbuf, int scount, MPI_Datatype stype, void *rbuf, int rcount, MPI_Datatype rtype, MPI_Comm comm).



Мал. 8.6. Схема передачі даних для підпрограми MPI_Allgather

4. Векторні підпрограми розподілу даних

Підпрограми MPI_Scatterv і MPi_Gatnerv є розширеними ("векторними") версіями підпрограм MPI_scatter і MPI_Gather. Вони дозволяють пересилати різним процесам (або збирати від них) різну кількість елементів даних. Завдяки аргументу displs, який задає відстань між елементами, елементи які пересилаються можуть не розташовуватися безперервно в пам'яті головного процесу. Це може виявитися корисним, наприклад, при пересиланні частин масивів.

Векторна підпрограма розподілу має наступні параметри:

int MPI_Scatterv(void *sendbuf, int *sendcounts, int *displs, MPI_Datatype sendtype, void *rcvbuf, int rcvcount,MPI_Datatype rcvtype, int root, MPI_Comm comm)

Вхідні параметри:

- sendbuf - адреса буфера передачі;
- send counts - цілочисельний одновимірний масив, що містить кількість елементів, переданих кожному процесу (індекс дорівнює рангу адресата). Його довжина дорівнює кількості процесів в комунікаторі;

- displs — цілочисельний масив, довжина якого дорівнює кількості процесів в комунікаторі. Елемент з індексом і задає зсув відносно початку буфера передачі. Ранг адресата дорівнює значенню індексу і;
- sendtype — тип даних в буфері передачі;
- rcvcount — кількість елементів у буфері прийому;
- rcvtype — тип даних в буфері прийому;
- root — ранг передавального процесу;
- comm — комунікатор.

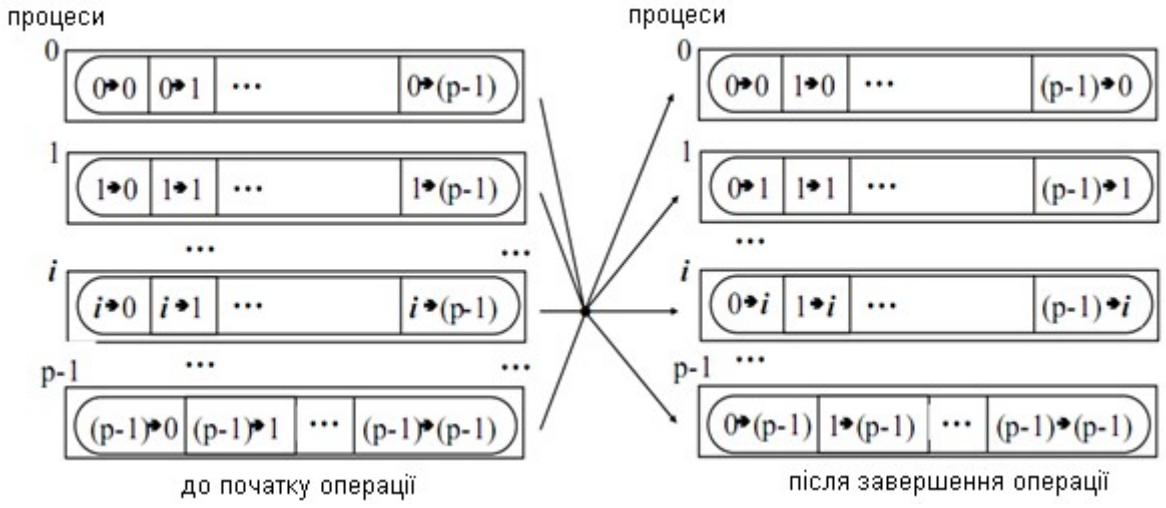
Вихідним є адреса буфера прийому rcvbuf.

Підпрограма MPI_Gatherv використовується для збору даних від всіх процесів в заданому комунікаторі та запису їх в буфер прийому із зазначеним зміщенням:
int MPI_Gatherv(void *sendbuf, int sendcount, MPI_Datatype sendtype, void *recvbuf, int *recvcounts, int *displs, MPI_Datatype recvtype, introot, MPI_Commcomm)

Список параметрів у цієї підпрограми схожий на список параметрів підпрограми MPI_scatterv.

Виконання загального варіанту операції збору даних, коли розміри повідомлень, що передаються процесами, можуть бути різні, забезпечується за допомогою функції MPI_Allgathev.

5. Операції передачі даних від всіх процесів всім процесам



Мал. 8.7. Загальна схема операції передачі даних від всіх процесів всім процесам (повідомлення показуються позначеннями вигляду i=>j, де і і j є ранги передавальних і приймаючих процесів відповідно)

Кожен процес в комунікаторі передає дані з scount елементів кожному процесу (загальний розмір повідомлень, що відправляються, в процесах має бути рівний scount * p елементів, де p є кількість процесів в комунікаторі comm) і приймає повідомлення від кожного процесу.

Int MPI_Alltoall(void *sbuf,intscount,MPI_Datatypestype, void *rbuf, int rcount, MPI_Datatype rtype, MPI_Commcomm),

Де:

- sbuf, scount, stype- параметри повідомлень, що передаються
- rbuf, rcount, rtype- параметри повідомлень, що приймаються
- comm- коммунікатор, в рамках якого виконується передача даних.

Особливості:

1. Виклик функції MPI_Alltoall при виконанні операції загального обміну даними має бути виконаний в кожному процесі коммунікатора.

2. Варіант операції загального обміну даних, коли розміри передаваних процесами повідомлень можуть бути різні, забезпечується за допомогою функції MPI_Alltoallv.

У обмінах, що виконуються підпрограмами MPI_Allgather і MPI_Alltoall, немає головного процесу. Деталі відправки і прийому важливі для усіх процесів, що беруть участь в обміні. Підпрограма MPI_Allgather збирає дані від усіх процесів і розподіляє їх усім процесам (мал. 8.6). Дія цієї підпрограми рівносильна дії послідовності викликів підпрограми MPI_Gather, в кожному з яких як головний використовуються різні процеси. Прийом виконується в усіх завданнях. Буфер прийому послідовно заповнюється повідомленнями від усіх процесів-посилачів.

int MPI_Allgather(void *sendbuf, int sendcount, MPI_Datatype sendtype, void *rcvbuf, int rcvcount, MPI_Datatype rcvtype, MPI_Comm comm)

Вхідні параметри цієї підпрограми:

- sendbuf — початкова адреса буфера передачі;
- sendcount — кількість елементів у буфері передачі;
- sendtype — тип переданих даних;
- rcvcount — кількість елементів, отриманих від кожного процесу;
- rcvtype — тип даних у буфері прийому;
- comm — комунікатор.

Вихідним параметром є адреса буфера прийому (rcvbuf). Блок даних, переданий від i-го процесу, приймається кожним процесом і розміщується в i-м блоці буфера прийому recvbuf.

Підпрограма MPI_Alltoall пересилає дані за схемою "кожен - всім":

int MPI_Alltoall(void *sendbuf, int sendcount, MPI_Datatype sendtype, void *rcvbuf, int rcvcount, MPI_Datatype rcvtype, MPI_Comm comm)

Вхідні параметри:

- sendbuf — початкова адреса буфера передачі;
- sendcount — кількість елементів даних, що пересилаються кожному процесу;
- sendtype — тип даних у буфері передачі;
- rcvcount — кількість елементів даних, що приймаються від кожного процесу;
- rcvtype — тип даних, що приймаються;
- comm — комунікатор.

Вихідний параметр - адреса буфера прийому rcvbuf.

Векторними версіями MPI _ Allgather і MPI _ Alltoall є підпрограми MPI _ Allgatherv і MPI _ Alltoallv.

Підпрограма MPI _ Allgatherv збирає дані від усіх процесів і пересилає їх усім процесам:

int MPI_Allgatherv(void *sendbuf, int sendcount, MPI_Datatype sendtype, void *rcvbuf, int *rcvcounts, int *displs, MPI_Datatype rcvtype, MPI_Comm comm)

Її параметри співпадають з параметрами підпрограми MPI _ Allgatner, за винятком додаткового вхідного параметра displs. Це цілочисельний одновимірний масив, кількість елементів в якому дорівнює кількості процесів в комунікаторові. Елемент масиву з індексом і задає зміщення відносно початку буфера прийому recvbuf, в якому розташовуються дані, що приймаються від процесу і. Блок даних, переданий від j- го процесу, приймається кожним процесом і розміщується в j -му блоці буфера прийому.

Підпрограма MPI_Alltoallv пересилає дані від усіх процесів усім процесам зі зміщенням:

int MPI_Alltoallv(void *sendbuf, int *sendcounts, int *sdispls, MPI_Datatype sendtype, void *rcvbuf, int *rcvcounts, int *rdispls, MPI_Datatype rcvtype, MPI_Comm comm)

Її параметри аналогічні параметрам підпрограми MPI _ Alltoall, окрім двох додаткових параметрів:

- sdispls- цілочисельний масив, кількість елементів в якому дорівнює кількості процесів в комунікаторі. Елемент j задає зміщення відносно початку буфера, з якого дані передаються у'-му процесу.

- rdispls - цілочисельний масив, кількість елементів в якому дорівнює кількості процесів в комунікаторі. Елемент і задає зміщення відносно початку буфера, в який приймається повідомлення від і- го процесу.

Табл. 8.1. Підпрограми розподілу і збору даних

Підпрограма	Короткий опис
MPI_Allgather	Збирає дані від всіх процесів і пересилає їх усім процесам
MPI Allgatherv	Збирає дані від всіх процесів і пересилає їх усім процесам ("векторний" варіант підпрограми MPI_Allgather)
MPI Allreduce	Збирає дані від всіх процесів, виконує операцію приведення, і результат розподіляє всім процесам
MPI_Alltoall	Пересилає дані від всіх процесів всім процесам
MPI Alltoallv	Пересилає дані від всіх процесів всім 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_Allgather	Збирає дані від всіх процесів і пересилає їх усім процесам
MPI Allgatherv	Збирає дані від всіх процесів і пересилає їх усім процесам ("векторний" варіант підпрограми MPI_Allgather)
MPI Allreduce	Збирає дані від всіх процесів, виконує операцію приведення, і результат розподіляє всім процесам
MPI_Alltoall	Пересилає дані від всіх процесів всім процесам

6. Операції приведення і сканування

Операції приведення і сканування належать до категорії глобальних обчислень. У глобальній операції приведення до даних від усіх процесів із заданого комунікатора застосовується операція MPI _ Reduce.

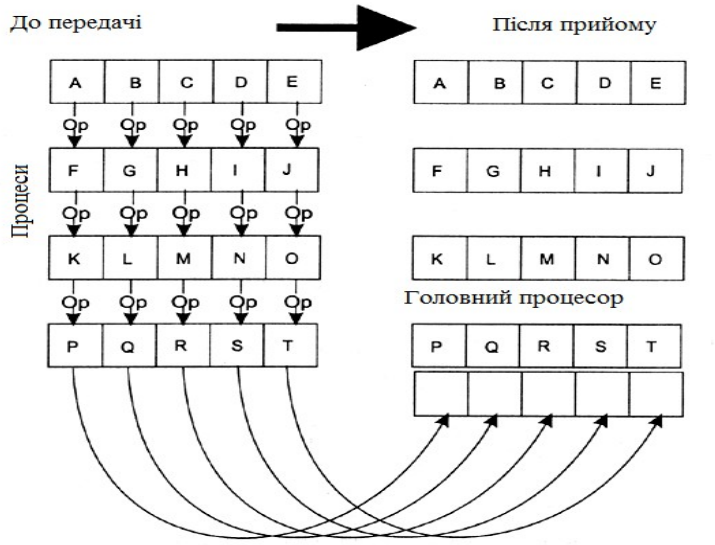
Аргументом операції приведення є масив даних - по одному елементу від кожного процесу. Результат такої операції - єдине значення.

У підпрограмах глобальних обчислень можливі наступні функції, що передається в підпрограму:

зумовлена функція MPI, наприклад MPI _ SUM;

призначена для користувача функція;

обробник призначеної для користувача функції, який створюється підпрограмою Mpi_op_create.



Мал. 8.8. Операція приведення

Три версії операції приведення повертають результат:

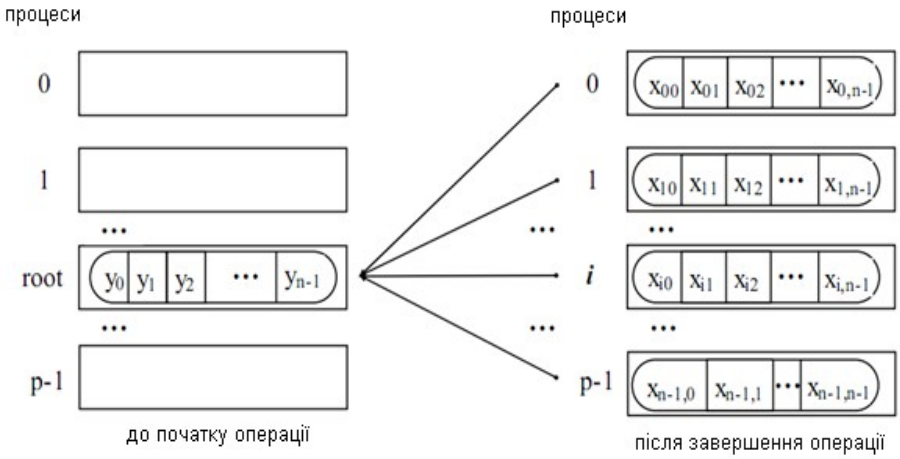
- одному процесу;
- усім процесам;
- розподіляють вектор результатів між усіма процесами.

Операція приведення, результат якої передається одному процесу, виконується при виклику підпрограми MPI _ Reduce:

int MPI_Reduce(void *buf, void *result, int count, MPI_Datatype datatype, MPI_Op op, int root, MPI_Comm comm)

Вхідні параметри підпрограми MPI _ Reduce:

- buf - адреса буфера передачі;
- count - кількість елементів у буфері передачі;
- datatype - тип даних у буфері передачі;
- op - операція приведення;
- root - ранг головного процесу;
- comm - комунікатор.



Мал. 8.9. Глобальна операція приведення

Підпрограма MPI_Reduce застосовує операцію приведення до операндів з buf, а результат кожної операції поміщається у буфер результату result. MPI_Reduce повинна викликатися усіма процесами в комунікаторові comm, а аргументи count, datatype і op в цих викликах повинні співпадати.

Функція приведення (op) не повертає код помилки, тому при виникненні аварійної ситуації або завершується робота усієї програми, або помилка мовчазно ігнорується.

У MPI є 12 зумовлених операцій приведення:

Табл. 8.2. Список зумовлених операцій приведення MPI

Операція	**Опис**
MPI_MAX	Визначення максимальних значень елементів одновимірних масивів цілого або речового типу
MPI_MIN	Визначення мінімальних значень елементів одновимірних масивів цілого або речового типу
MPI_SUM	Обчислення суми елементів одновимірних масивів цілого, речового або комплексного типу
MPI_PROD	Обчислення поелементного добутку одновимірних масивів цілого, речового або комплексного типу
MPI LAND	Логічне " І"
MPI BAND	Бітове " І"
MPI_LOR	Логічне " АБО"
MPI_BOR	Бітове " АБО"
MPI_LXOR	Логічне виключаюче "АБО"

Операція	Опис
MPI_BXOR	Бітове виключаюче "АБО"
MPI MAXLOC	Максимальні значення елементів одновимірних масивів та їх індекси
MPI MINLOC	Мінімальні значення елементів одновимірних масивів та їх індекси

Програміст може визначити і власні глобальні операції приведення. Це робиться за допомогою підпрограми MPI_Op_create**:**

int MPI_Op_create(MPI_User_function *function, int commute, MPI_Op *op)

Вхідними параметрами цієї підпрограми є:

- function — функція користувача;
- commute — прапор, яким присвоюється значення "істина", якщо операція коммутативна (тобто її результат не залежить від порядку операндів).

Опис типу функції користувача виглядає наступним чином:

typedef void (MPI_User_function)

(void*a,* void *b, int *len, MPI_Datatype *dtype)

Тут операція визначається так:

b[I] = *a [I]* op b[I] ДЛЯ 1 = 0, ..., len - *1.*

Після того, як функція користувача зіграла свою роль, її слід анулювати за допомогою підпрограми MPI_op_free:

int MPI_Op_free(MPI_Op *op)

По завершенні виклику змінній op присвоюється значення MPI_OP_NULL.

У підпрограми MPI_Reduce є варіанти.

Підпрограма MPI_Reduce_scatter збирає і розподіляє дані:

int MPI_Reduce_scatter(void *sendbuf, void *rcvbuf, int *rcvcounts, MPI_Datatype datatype, MPI_Op op, MPI_Comm comm)

Її вхідні параметри:

- sendbuf — стартова адреса буфера прийому;
- rcvcounts — цілочисельний одновимірний масив, який задає кількість елементів в результуючому масиві, що розподіляється кожному процесу. Цей масив має бути однаковим в усіх процесах, що викликають цю підпрограму;
- datatype — тип даних у буфері прийому;
- op — операція;
- comm — комунікатор.

Вихідний параметр - стартова адреса буфера прийому rcvbuf. Особливістю цієї підпрограми є те, що кожне завдання отримує не увесь результуючий масив, а його частина.

Підпрограма MPI _ Allreduce збирає дані від усіх процесів і зберігає результат операції приведення в результуючому буфері кожного процесу:

int MPI_Allreduce(void *sendbuf, void *rcvbuf, int count, MPI_Datatype datatype, MPI_Op op, MPI_Comm comm)

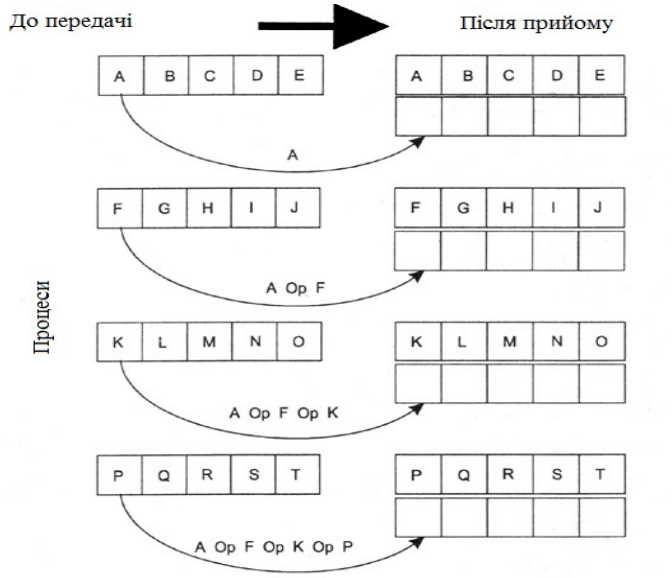
Її вхідні параметри:

- sendbuf — початкова адреса буфера передачі;
- count — кількість елементів у буфері передачі;
- datatype — тип переданих даних;
- op — операція приведення;
- comm — комунікатор.

Вихідним параметром є стартова адреса буфера прийому rcvbuf. При аварійному завершенні підпрограма може повертати код помилки MPI_ERR_OP (некоректна операція). Це відбувається, якщо застосовується операція, яка не є зумовленою і яка не створена попереднім викликом підпрограми MPI_op_create.

Операції сканування (частковій редукції) виконуються за допомогою виклику підпрограми MPI_Scan:

int MPI_Scan(void *sendbuf, void *rcvbuf, int count, MPI_Datatype datatype, MPI_Op op, MPI_Comm comm)



Мал. 8.10. Загальна схема операції сканування

Її вхідні параметри:

- sendbuf — початкова адреса буфера передачі;
- ount — кількість елементів у вхідному буфері;
- datatype — тип даних у вхідному буфері;
- op — операція;
- comm — комунікатор.

Вихідним параметром є стартова адреса буфера прийому rcvbuf.

У підпрограмі MPI_Reduce можна використати тільки зумовлені типи MPI.

Усі операції є асоціативними і комутативними (тобто операнди можуть по-різному групуватися, а їх порядок не має значення).

Підпрограма MPI_scan схожа на підпрограму MPI_Allreduce в тому відношенні, що кожне завдання отримує результуючий масив. Відмінність полягає в тому, що вміст масиву-результату в завданні і є результатом виконання операції над масивами із завдань з номерами від 0 до і включно.

У кожному процесі можна використати різні призначені для користувача операції. У MPI не визначено, які операції і над якими операндами застосовуватимуться в цьому випадку. У буферах передачі допускається перекриття типів, у буферах прийому це може привести до непередбачуваних результатів.

Лістинг 8.3. Приклад використання операції редукції

```
#include "mpi.h"
finclude<stdio.h>
int main(int argc,char *argv[])
{
int rayrank, i;
int count = 5, root = 1;
MPI_Group MPI_GROUP_WORLD, subgroup;
int ranks[4] = {1, 3, 5, 7};
MPI_Comm subcomm;
int sendbuf[5] = {1, 2, 3, 4, 5};
int recvbuf[5];
MPI_Init(sargc, sargv);
MPI_Comm_group(MPI_COMM_WORLD, &MPI_GROUP_WORLD);
MPI_Group_incl(MPI_GROUP_WORLD, 4, ranks, Ssubgroup);
MPI_Group_rank(subgroup, Smyrank);
MPI_Coram_create(MPI_COMM_WORLD, subgroup, ssubcomm);
if(myrank != MPI_UNDEFINED)
```

```
{
MPI_Reduce(&sendbuf, &recvbuf, count, MPI_INT, MPI_SUM, root, subcomm);
if(myrank == root) { printf("Reduced values");
for(i = 0; i < count; i++){ printf(" %i ", recvbuf[i]);}
printf ("\n") ;
MPI_Comm_free (&subcomm) ;
MPI_Group_free (&MPI_GROUP_WORLD)
MPI_Group_free (&subgroup) ; }
MPI_Finalize() ;
return 0;
}
```

У цій програмі спочатку створюється підгрупа, що складається з процесів з рангами 1, 3, 5 і 7 (звідси витікає, що запускати її на виконання потрібно не менше чим у восьми процесах), і комунікатор, що відповідає їй. Редукція виконується тільки процесами з цієї групи. У кінці програми усі створені в процесі її роботи описувачі мають бути видалені.