Основы параллельного программирования с использованием MPI

Лекция 6

Немнюгин Сергей Андреевич

Санкт-Петербургский государственный университет

физический факультет

кафедра вычислительной физики

Интернет-Университет
Суперкомпьютерных Технологий

High-Performance Computing University

Лекция 6

Аннотация

В лекции рассматриваются средства организации групп процессов и соответствующих им коммуникаторов. Речь идёт об интра- и интеркоммуникаторах. Обсуждаются основные операции управления группами и коммуникаторами. Разбираются примеры использования двухточечных и коллективных обменов в производных группах процессов, а также обмена между группами процессов.

План лекции

- Группы процессов и коммуникаторы
- Интракоммуникаторы.
- Управление группами процессов.
- Управление коммуникаторами.
- Интеркоммуникаторы и организация обменов между группами процессов.

Группы процессов и коммуникаторы

Группы процессов и коммуникаторы

В MPI имеются средства создания и преобразования коммуникаторов, которые дают возможность программисту в дополнение к стандартным предопределенным объектам создавать свои собственные. Это позволяет использовать разнообразные схемы взаимодействия процессов.

Используя средства MPI, можно, например, создать новый коммуникатор, содержащий те же процессы, что и исходный, но с новым контекстом (новыми свойствами).

Обмен возможен только в рамках одного контекста, обмены в разных коммуникаторах происходят независимо.

Группы процессов и коммуникаторы

Новый коммуникатор можно передать в качестве аргумента библиотечной подпрограмме, как в следующем примере:

```
…
MPI_Comm_dup(comm, newcomm, ierr)
gauss_parallel(newcomm, a, b)
MPI_Comm_free(newcomm, ierr)
…
```

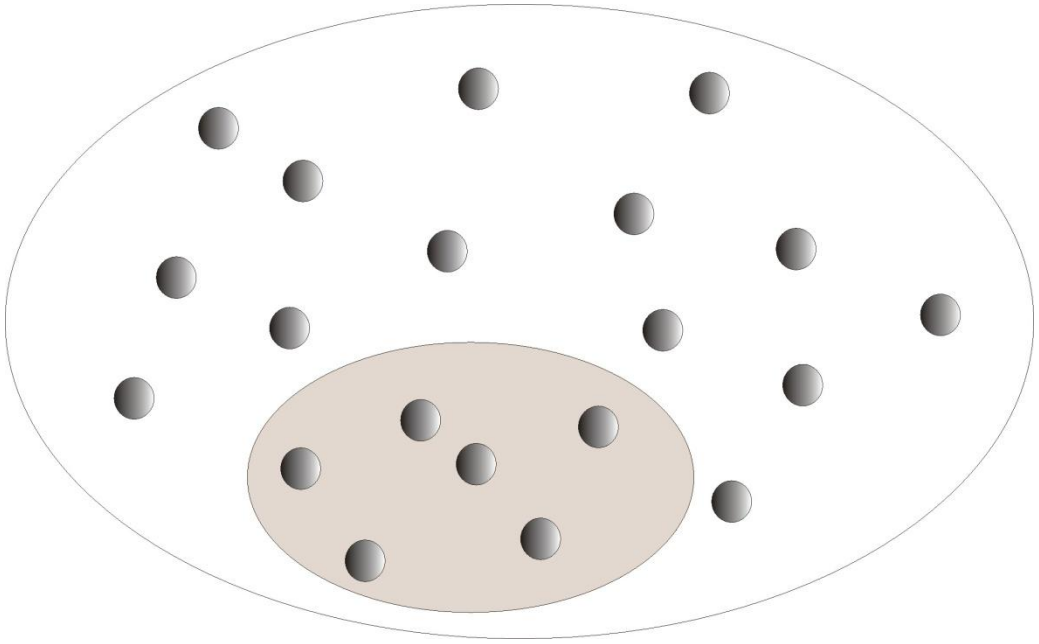
Здесь сначала создается новый коммуникатор (подпрограмма `MPI_Comm_dup`).

Подпрограмма `gauss_parallel` использует коммуникатор `newcomm`, так, что все обмены в ней выполняются независимо от других операций обмена.

Группы процессов и коммуникаторы

Группы процессов

Группой называют упорядоченное множество процессов. Каждому процессу в группе сопоставлен свой ранг. Операции с группами могут выполняться отдельно от операций с коммуникаторами, но в операциях обмена используются только коммуникаторы.
В MPI имеется предопределенная пустая группа `MPI_GROUP_EMPTY`.



Группы процессов и коммуникаторы

Коммуникаторы

Коммуникаторы бывают двух типов:

1. *интракоммуникаторы* —для операций внутри одной группы процессов;
2. *интеркоммуникаторы* —для обмена между двумя группами процессов.

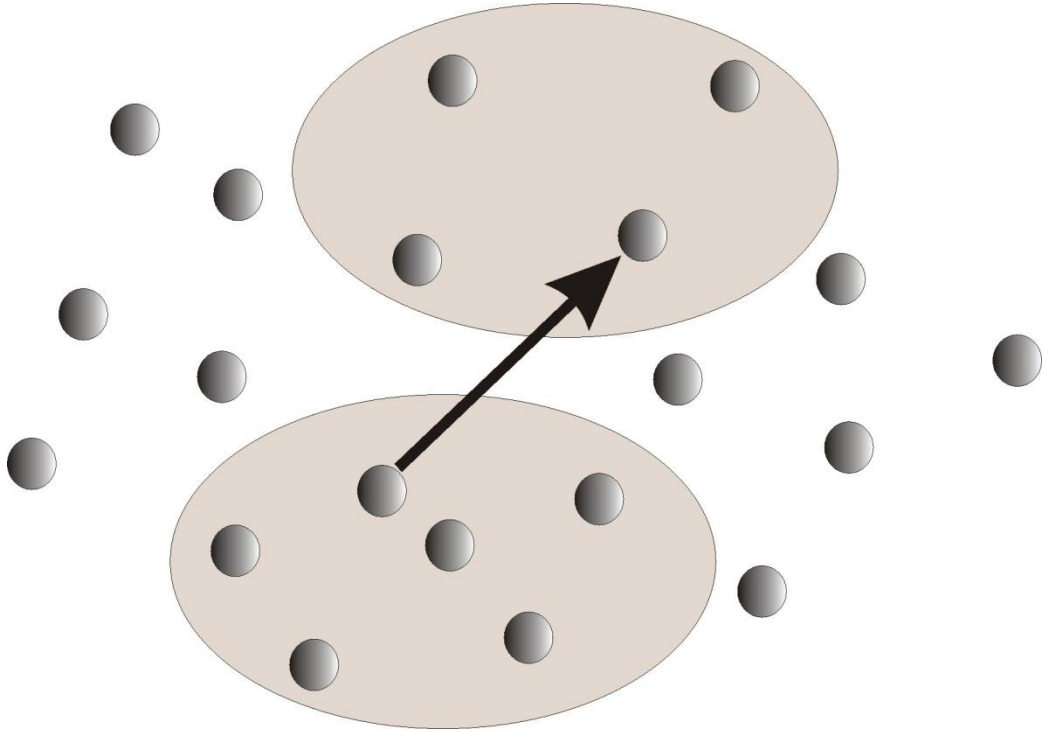
Интракоммуникатором является `MPI_COMM_WORLD`.
В MPI-программах чаще используются интракоммуникаторы.
Интракоммуникатор включает: экземпляр группы, контекст обмена для всех его видов, а также, возможно, виртуальную топологию и другие атрибуты. Контекст обеспечивает возможность создания изолированных друг от друга, а потому безопасных областей взаимодействия. Система сама управляет их разделением. Контекст играет роль дополнительного тега, который дифференцирует сообщения.

Группы процессов и коммуникаторы

MPI поддерживает обмены между двумя непересекающимися группами процессов. Если параллельная программа состоит из нескольких параллельных модулей, удобно разрешить одному модулю обмениваться сообщениями с другим, используя для адресации локальные по отношению ко второму модулю ранги. Такой подход удобен, например, при программировании параллельных клиент-серверных приложений. Интеробмены реализуются с помощью *интеркоммуникаторов*, которые объединяют две группы процессов общим контекстом.

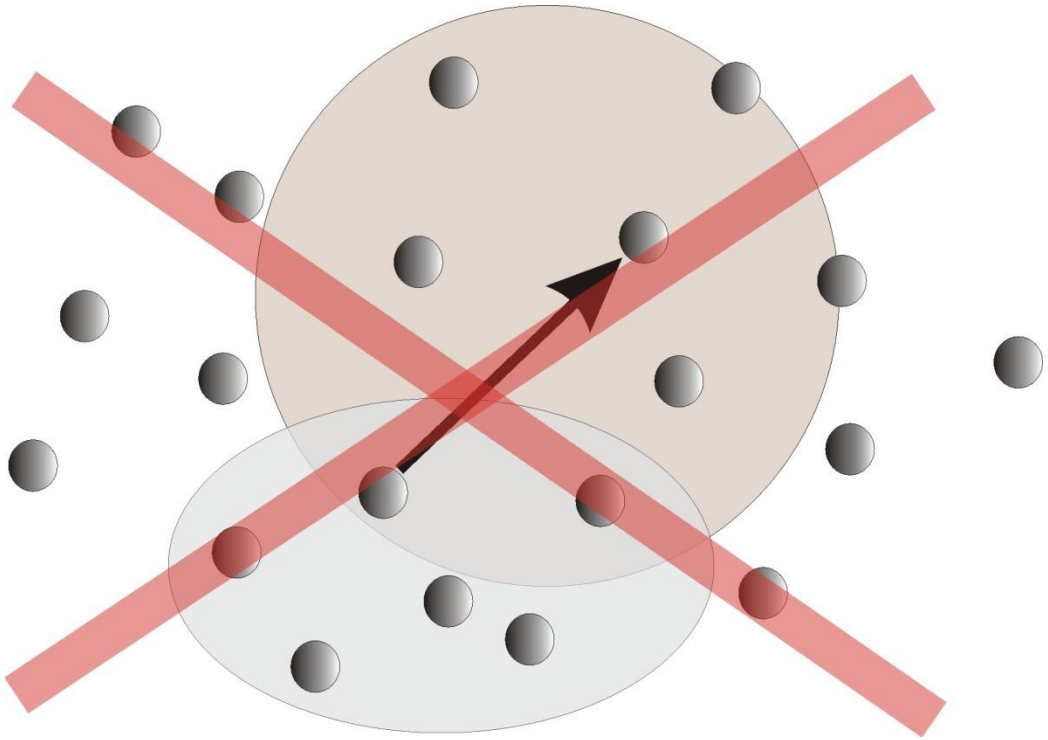
Группы процессов и коммуникаторы

В контексте, связанном с интеркоммуникатором, передача сообщения в локальной группе всегда сопровождается его приемом в удаленной группе — это двухточечная операция (MPI-1). Группа, содержащая процесс, который инициирует операцию интеробмена, называется *локальной группой*. Группа, содержащая процесс-адресат, называется *удаленной группой*. Интеробмен задается парой: коммуникатор — ранг, при этом ранг задается относительно удаленной группы.



Группы процессов и коммуникаторы

Конструкторы интеркоммуникаторов являются блокирующими операциями. Во избежание тупиковых ситуаций локальная и удаленная группа не должны пересекаться, то есть они не должны содержать одинаковые процессы.



Группы процессов и коммуникаторы

Интеробмен характеризуется следующими свойствами:

- синтаксис двухточечных обменов одинаков в операциях обмена в рамках интра- и интеркоммуникаторов;
- адресат сообщения задается рангом процесса в удаленной группе;
- операции обмена с использованием интеркоммуникаторов не вступают в конфликт с обменами, использующими другой коммуникатор;
- интеркоммуникатор нельзя использовать для коллективных обменов (MPI-1);
- коммуникатор не может объединять свойства интер- и интракоммуникатора.

Возможности интеробменов расширены в MPI-2.
Интеркоммуникатор создается коллективным вызовом подпрограммы `MPI_Intercomm_create`.

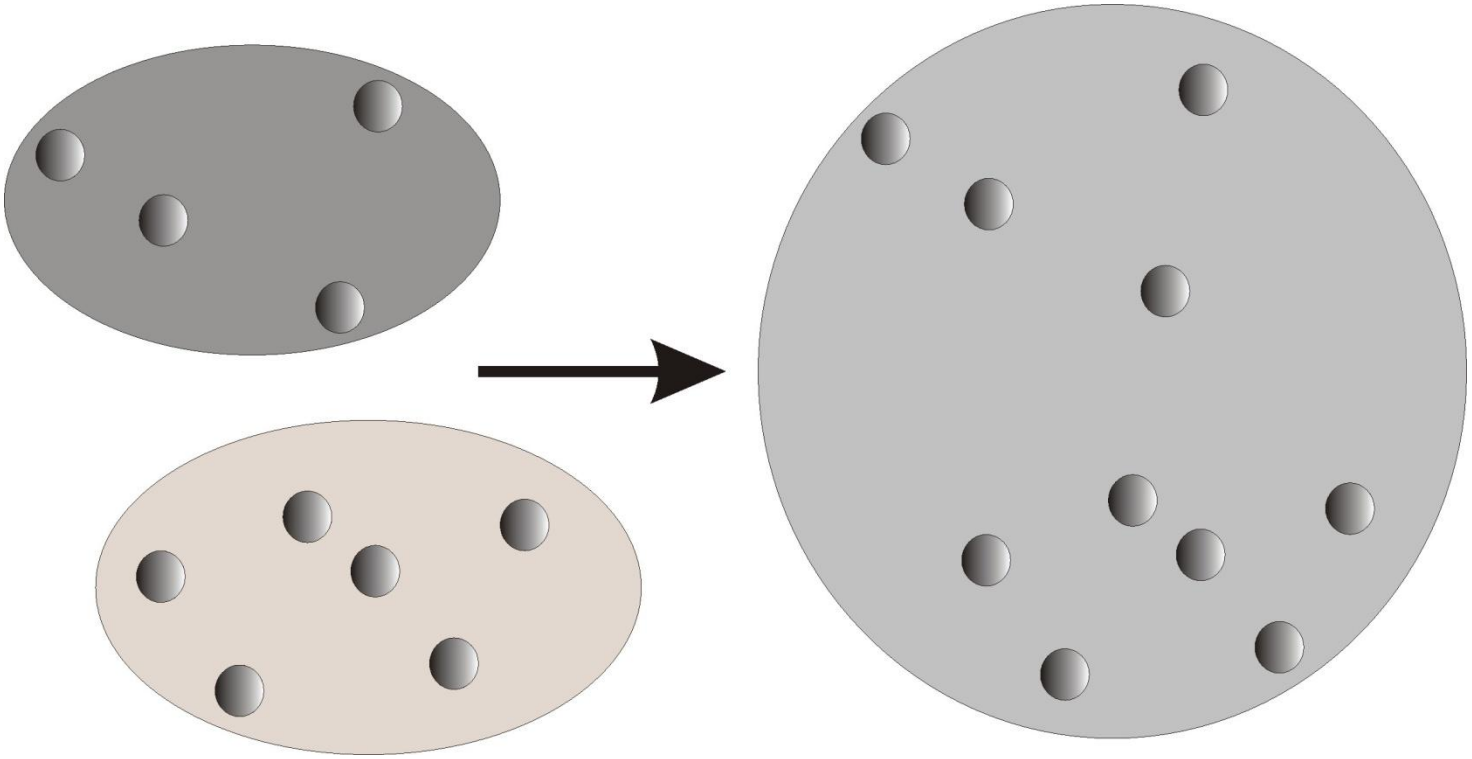
Создание групп процессов

Создание групп процессов

Создание групп процессов

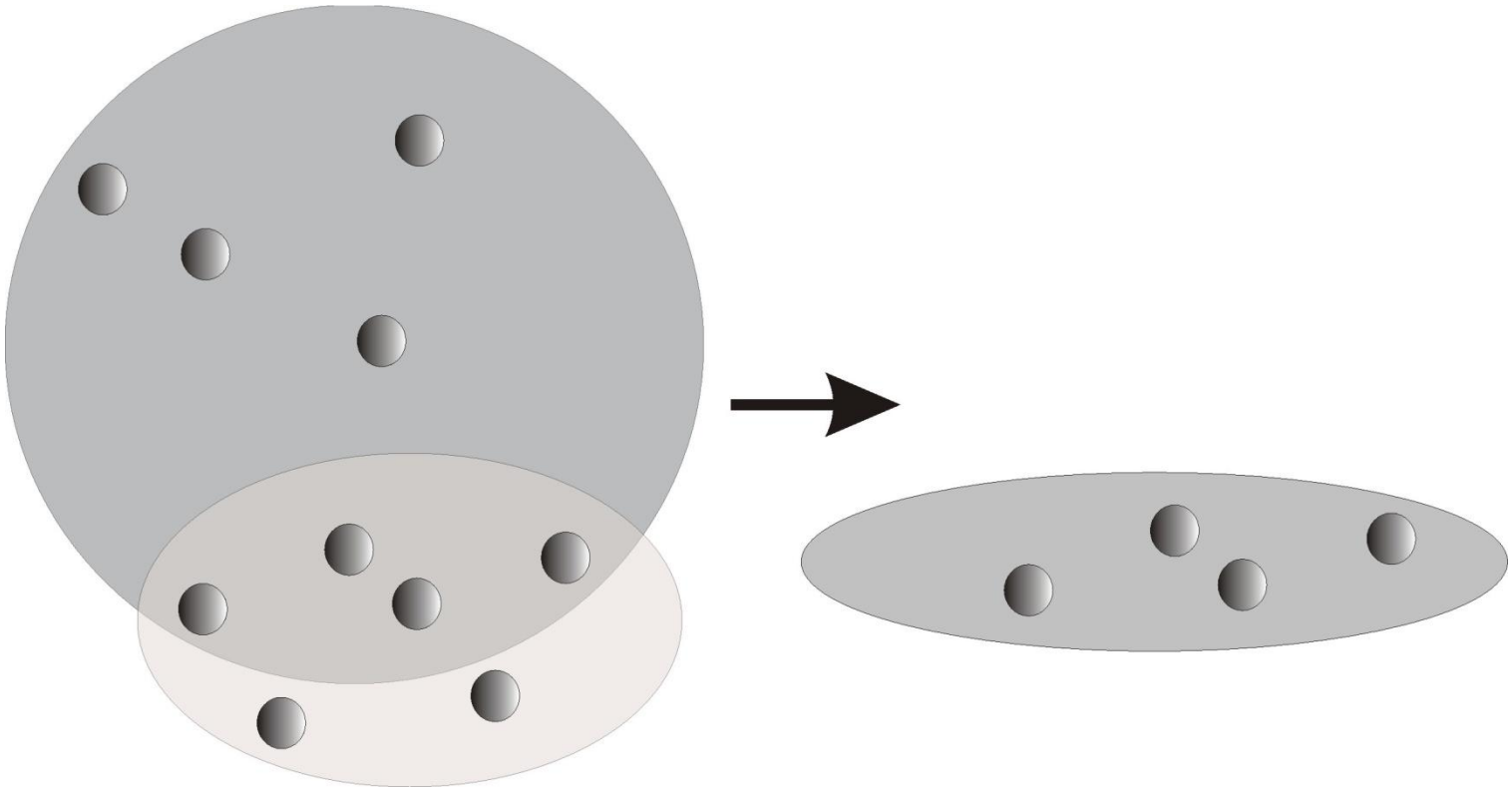
Созданию нового коммуникатора предшествует создание соответствующей группы процессов. Операции создания групп аналогичны математическим операциям над множествами:

- *объединение* —к процессам первой группы (`group1`) добавляются процессы второй группы (`group2`), не принадлежащие первой;



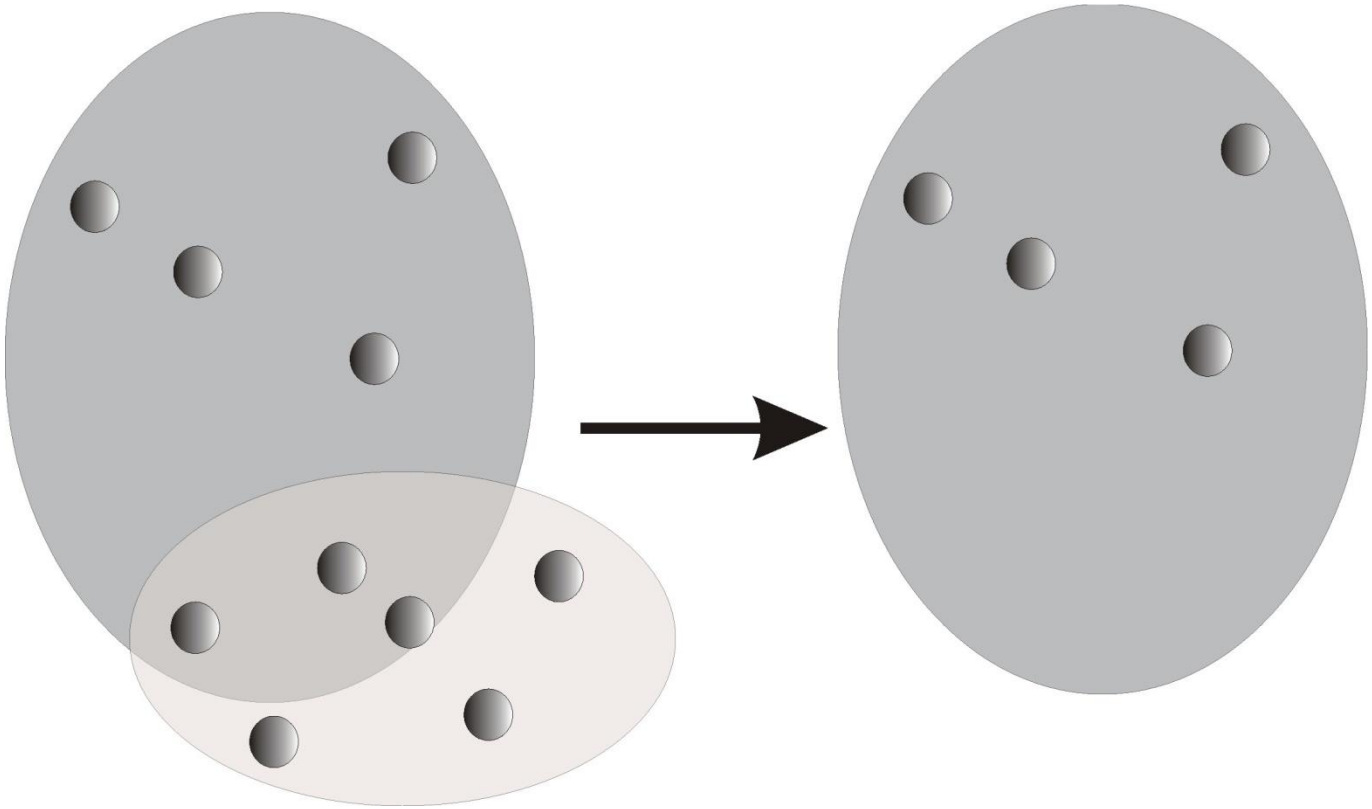
Создание групп процессов

- *пересечение* —в новую группу включаются все процессы, принадлежащие двум группам одновременно. Ранги им назначаются как в первой группе;



Создание групп процессов

- *разность* —в новую группу включаются все процессы первой группы, не входящие во вторую группу. Ранги назначаются как в первой группе.



Создание групп процессов

Новую группу можно создать только из уже существующих групп. Базовая группа, из которой формируются все другие группы, связана с коммуникатором `MPI_COMM_WORLD`. В подпрограммах создания групп, как правило, нельзя использовать пустой коммуникатор `MPI_COMM_NULL`.

Доступ к группе `group`, связанной с коммуникатором `comm` можно получить, обратившись к подпрограмме `MPI_Comm_group`:

```
int MPI_Comm_group(MPI_Comm comm, MPI_Group *group)

MPI_Comm_group(comm, group, ierr)
```

Ее выходным параметром является группа. Для выполнения операций с группой к ней сначала необходимо получить доступ.

Создание групп процессов

Подпрограмма `MPI_Group_incl` создает новую группу `newgroup` из `n` процессов, входящих в группу `oldgroup`. Ранги этих процессов содержатся в массиве `ranks`:

```
int MPI_Group_incl(MPI_Group oldgroup, int n, int *ranks,
MPI_Group *newgroup)

MPI_Group_incl(oldgroup, n, ranks, newgroup, ierr)
```

В новую группу войдут процессы с рангами `ranks[0], …, ranks[n − 1]`, причем рангу `i` в новой группе соответствует ранг `ranks[i]` в старой группе. При n = 0 создается пустая группа `MPI_GROUP_EMPTY`. С помощью данной подпрограммы можно не только создать новую группу, но и изменить порядок процессов в старой группе.

Создание групп процессов

Подпрограмма `MPI_Group_excl` создает группу `newgroup`, исключая из исходной группы (`group`) процессы с рангами `ranks[0], …, ranks[n – 1]`:

```
int MPI_Group_excl(MPI_Group oldgroup, int n, int *ranks,
MPI_Group *newgroup)

MPI_Group_excl(oldgroup, n, ranks, newgroup, ierr)
```

При n = 0 новая группа тождественна старой.

Создание групп процессов

Подпрограмма `MPI_Group_range_incl` создает группу `newgroup` из группы `group`, добавляя в нее `n` процессов, ранг которых указан в массиве `ranks`:

```
int MPI_Group_range_incl(MPI_Group oldgroup, int n, int
ranks[][3], MPI_Group *newgroup)

MPI_Group_range_incl(oldgroup, n, ranks, newgroup, ierr)
```

Массив `ranks` состоит из целочисленных триплетов вида (первый_1, последний_1, шаг_1), …, (первый_n, последний_n, шаг_n). В новую группу войдут процессы с рангами (по первой группе) первый_1, первый_1 + шаг_1, ….

Создание групп процессов

Подпрограмма `MPI_Group_range_excl` создает группу `newgroup` из группы `group`, исключая из нее `n` процессов, ранг которых указан в массиве `ranks`:

```
int MPI_Group_range_excl(MPI_Group group, int n, int
ranks[][3], MPI_Group *newgroup)

MPI_Group_range_excl(group, n, ranks, newgroup, ierr)
```

Массив `ranks` устроен так же, как аналогичный массив в подпрограмме `MPI_Group_range_incl`.

Создание групп процессов

Подпрограмма `MPI_Group_difference` создает новую группу (`newgroup`) из разности двух групп (`group1`) и (`group2`):

```
int MPI_Group_difference(MPI_Group group1, MPI_Group group2,
MPI_Group *newgroup)

MPI_Group_difference(group1, group2, newgroup, ierr)
```

Подпрограмма `MPI_Group_intersection` создает новую группу (`newgroup`) из пересечения групп `group1` и `group2`:

```
int MPI_Group_intersection(MPI_Group group1, MPI_Group group2,
MPI_Group *newgroup)

MPI_Group_intersection(group1, group2, newgroup, ierr)
```

Создание групп процессов

Подпрограмма `MPI_Group_union` создает группу (`newgroup`), объединяя группы `group1` и `group2`:

```
int MPI_Group_union(MPI_Group group1, MPI_Group group2,
MPI_Group *newgroup)

MPI_Group_union(group1, group2, newgroup, ierr)
```

Имеются и другие подпрограммы-конструкторы новых групп.

Создание групп процессов

Деструктор группы

Вызов подпрограммы `MPI_Group_free` уничтожает группу `group`:

```
int MPI_Group_free(MPI_Group *group)

MPI_Group_free(group, ierr)
```

Создание групп процессов

Получение информации о группе

Для определения количества процессов (`size`) в группе (`group`) используется подпрограмма `MPI_Group_size`:

```
int MPI_Group_size(MPI_Group group, int *size)

MPI_Group_size(group, size, ierr)
```

Создание групп процессов

Подпрограмма `MPI_Group_rank` возвращает ранг (`rank`) процесса в группе `group`:

```
int MPI_Group_rank(MPI_Group group, int *rank)

MPI_Group_rank(group, rank, ierr)
```

Если процесс не входит в указанную группу, возвращается значение `MPI_UNDEFINED`.

Создание групп процессов

Процесс может входить в несколько групп. Подпрограмма `MPI_Group_translate_ranks` преобразует ранг процесса в одной группе в его ранг в другой группе:

```
int MPI_Group_translate_ranks(MPI_Group group1, int n, int
*ranks1, MPI_Group group2, int *ranks2)

MPI_Group_translate_ranks(group1, n, ranks1, group2, ranks2,
ierr)
```

Эта функция используется для определения относительной нумерации одних и тех же процессов в двух разных группах.

Создание групп процессов

Подпрограмма `MPI_Group_compare` используется для сравнения групп `group1` и `group2`:

```
int MPI_Group_compare(MPI_Group group1, MPI_Group group2, int
*result)

MPI_Group_compare(group1, group2, result, ierr)
```

Результат выполнения этой подпрограммы:

- Если группы полностью совпадают, возвращается значение `MPI_IDENT`.
- Если члены обеих групп одинаковы, но их ранги отличаются, результатом будет значение `MPI_SIMILAR`.
- Если группы различны, результатом будет `MPI_UNEQUAL`.

Управление коммуникаторами

Управление коммуникаторами

Создание коммуникатора —коллективная операция и соответствующая подпрограмма должна вызываться всеми процессами коммуникатора. Подпрограмма `MPI_Comm_dup` дублирует уже существующий коммуникатор `oldcomm`:

```
int MPI_Comm_dup(MPI_Comm oldcomm, MPI_Comm *newcomm)

MPI_Comm_dup(oldcomm, newcomm, ierr)
```

В результате вызова создается новый коммуникатор (`newcomm`) с той же группой процессов, с теми же атрибутами, но с другим контекстом. Подпрограмма может применяться как к интра-, так и к интеркоммуникаторам.

Управление коммуникаторами

Подпрограмма `MPI_Comm_create` создает новый коммуникатор (`newcomm`) из подмножества процессов (`group`) другого коммуникатора (`oldcomm`):

```
int MPI_Comm_create(MPI_Comm oldcomm, MPI_Group group, MPI_Comm
*newcomm)

MPI_Comm_create(oldcomm, group, newcomm, ierr)
```

Вызов этой подпрограммы должны выполнить все процессы из старого коммуникатора, даже если они не входят в группу `group`, с одинаковыми аргументами. Данная операция применяется только к интра-коммуникаторам. Она позволяет выделять подмножества процессов со своими областями взаимодействия, если, например, требуется уменьшить «зернистость» параллельной программы. Побочным эффектом применения подпрограммы `MPI_Comm_create` является синхронизация процессов. Если одновременно создаются несколько коммуникаторов, они должны создаваться в одной последовательности всеми процессами.

Управление коммуникаторами

Пример создания коммуникатора

```
#include "mpi.h"
#include <stdio.h>
int main(int argc,char *argv[])
{
 char message[24];
 MPI_Group MPI_GROUP_WORLD;
 MPI_Group group;
 MPI_Comm fcomm;
 int size, q, proc;
 int* process_ranks;
 int rank, rank_in_group;
 MPI_Status status;

 MPI_Init(&argc, &argv);
 MPI_Comm_size(MPI_COMM_WORLD, &size);
 MPI_Comm_rank(MPI_COMM_WORLD, &rank);
```

Управление коммуникаторами

```
 printf("New group contains processes:");
 q = size – 1;
 process_ranks = (int*) malloc(q*sizeof(int));
 for (proc = 0; proc < q; proc++)
{
 process_ranks[proc] = proc;
 printf("%i ", process_ranks[proc]);
 }
 printf("\n");
 MPI_Comm_group(MPI_COMM_WORLD, &MPI_GROUP_WORLD);
 MPI_Group_incl(MPI_GROUP_WORLD, q, process_ranks, &group);
 MPI_Comm_create(MPI_COMM_WORLD, group, &fcomm);
 if (fcomm != MPI_COMM_NULL) {
 MPI_Comm_group(fcomm, &group);
 MPI_Comm_rank(fcomm, &rank_in_group);
```

Управление коммуникаторами

```
 if (rank_in_group == 0) {
 strcpy(message, "Hi, Parallel Programmer!");
 MPI_Bcast(&message, 25, MPI_BYTE, 0, fcomm);
 printf("0 send: %s\n", message);
 }
else
{
 MPI_Bcast(&message, 25, MPI_BYTE, 0, fcomm);
 printf("%i received: %s\n", rank_in_group, message);
}
 MPI_Comm_free(&fcomm);
 MPI_Group_free(&group);
}
 MPI_Finalize();
 return 0;
}
```

Управление коммуникаторами

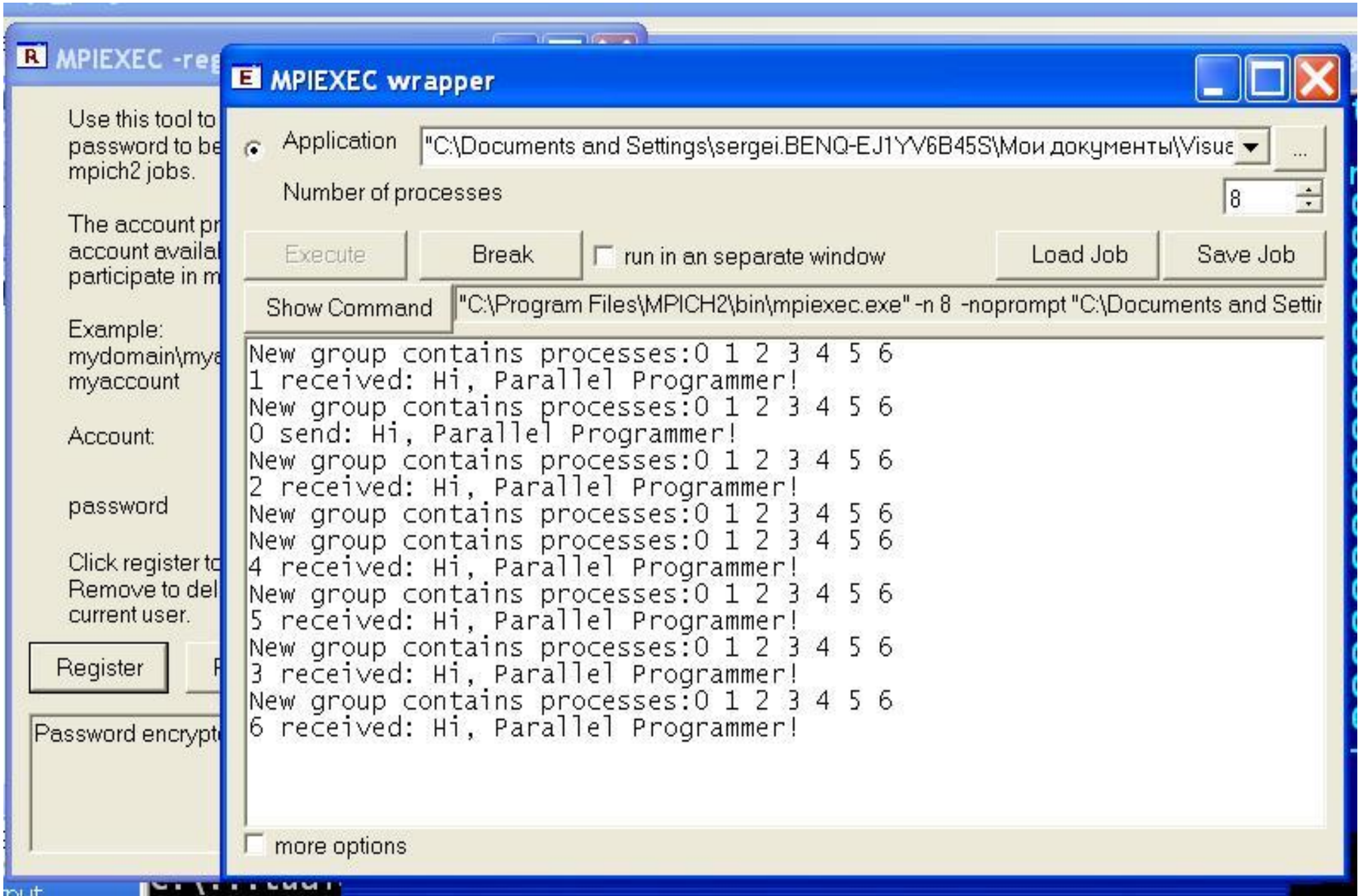
Эта программа работает следующим образом. Пусть в коммуникатор `MPI_COMM_WORLD` входят `p` процессов. Сначала создается список процессов, которые будут входить в область взаимодействия нового коммуникатора. Затем создается группа, состоящая из этих процессов. Для этого требуются две операции. Первая определяет группу, связанную с коммуникатором `MPI_COMM_WORLD`. Новая группа создается вызовом подпрограммы `MPI_Group_incl`. Затем создается новый коммуникатор. Для этого используется подпрограмма `MPI_Comm_create`. Новый коммуникатор —`fcomm`. В результате всех этих действий все процессы, входящие в коммуникатор `fcomm`, смогут выполнять операции коллективного обмена, но только между собой.

Результат выполнения программы:

Управление коммуникаторами

```
[nemnugin@pd00 ~]$ mpiexec -n 8 ./a.out
New group contains processes:0 1 2 3 4 5 6
New group contains processes:0 1 2 3 4 5 6
New group contains processes:0 1 2 3 4 5 6
New group contains processes:0 1 2 3 4 5 6
New group contains processes:0 1 2 3 4 5 6
New group contains processes:0 1 2 3 4 5 6
New group contains processes:0 1 2 3 4 5 6
New group contains processes:0 1 2 3 4 5 6
New group contains processes:0 1 2 3 4 5 6
0 send: Hi, Parallel Programmer!
2 received: Hi, Parallel Programmer!
1 received: Hi, Parallel Programmer!
6 received: Hi, Parallel Programmer!
3 received: Hi, Parallel Programmer!
5 received: Hi, Parallel Programmer!
4 received: Hi, Parallel Programmer!
[nemnugin@pd00 ~]$
```

Управление коммуникаторами



Управление коммуникаторами

Подпрограмма `MPI_Comm_split` позволяет создать несколько коммуникаторов сразу:

```
int MPI_Comm_split(MPI_Comm oldcomm, int split, int rank,
MPI_Comm* newcomm)

MPI_Comm_split(oldcomm, split, rank, newcomm, ierr)
```

Группа процессов, связанных с коммуникатором `oldcomm`, разбивается на непересекающиеся подгруппы, по одной для каждого значения аргумента `split`. Процессы с одинаковым значением `split` образуют новую группу. Ранг в новой группе определяется значением `rank`.

Управление коммуникаторами

Если процессы A и B вызывают `MPI_Comm_split` с одинаковым значением `split`, а аргумент `rank`, переданный процессом A, меньше, чем аргумент, переданный процессом B, ранг A в группе, соответствующей новому коммуникатору, будет меньше ранга процесса B. Если же в вызовах используется одинаковое значение `rank`, система присвоит ранги произвольно. Для каждой подгруппы создается собственный коммуникатор `newcomm`.

`MPI_Comm_split` — коллективная подпрограмма, ее должны вызвать все процессы из старого коммуникатора, даже если они не войдут в новый коммуникатор. Для этого в качестве аргумента `split` в подпрограмму передается предопределенная константа `MPI_UNDEFINED`. Соответствующие процессы вернут в качестве нового коммуникатора значение `MPI_COMM_NULL`. Новые коммуникаторы, созданные подпрограммой `MPI_Comm_split`, не пересекаются.

Управление коммуникаторами

В следующем примере создаются три новых коммуникатора (если исходный коммуникатор `comm` содержит не менее трех процессов):

```
MPI_Comm comm, newcomm;
int rank, split;
MPI_Comm_rank(comm, &rank);
split = rank%3;
MPI_Comm_split(comm, split, rank, &newcomm);
```

Если количество процессов 9, каждый новый коммуникатор будет содержать 3 процесса. Это можно использовать, например, для того, чтобы расщепить двумерную решетку 3x3, с каждым узлом которой связан один процесс, на 3 одномерных подрешетки.

Управление коммуникаторами

В следующем примере процессы разбиваются на две группы. Одна содержит процессы с чётными рангами, а другая – с нечётными.

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

void main(int argc, char *argv[])
{
   int num, p;
   int Neven, Nodd, members[6], even_rank, odd_rank;
   MPI_Group group_world, even_group, odd_group;

   MPI_Init(&argc, &argv);
   MPI_Comm_rank(MPI_COMM_WORLD, &num);
   MPI_Comm_size(MPI_COMM_WORLD, &p);
```

Управление коммуникаторами

```
Neven = (p + 1)/2;
Nodd = p - Neven;
members[0] = 2;
members[1] = 0;
members[2] = 4;
MPI_Comm_group(MPI_COMM_WORLD, &group_world);
MPI_Group_incl(group_world, Neven, members, &even_group);
MPI_Group_excl(group_world, Neven, members,  &odd_group);
MPI_Barrier(MPI_COMM_WORLD);
if(num == 0) {
 printf("Number of processes is %d\n", p);
 printf("Number of odd processes is %d\n", Nodd);
 printf("Number of even processes is %d\n", Neven);
 printf("members[0] is assigned rank %d\n", members[0]);
 printf("members[1] is assigned rank %d\n", members[1]);
 printf("members[2] is assigned rank %d\n", members[2]);
 printf("\n");
 printf("     num     even      odd\n");
}
```

Управление коммуникаторами

```
    MPI_Barrier(MPI_COMM_WORLD);
    MPI_Group_rank(even_group, &even_rank);
    MPI_Group_rank( odd_group,  &odd_rank);
    printf("%8d %8d %8d\n",num, even_rank, odd_rank);
    MPI_Finalize();
}
```

Управление коммуникаторами

Результат выполнения:

```
[nemnugin@pd00 ~]$ mpiexec -n 4 ./a.out
Number of processes is 4
Number of odd processes is 2
Number of even processes is 2
members[0] is assigned rank 2
members[1] is assigned rank 0
members[2] is assigned rank 4

      Iam       even        odd
        0          1   -32766
        2          0   -32766
        1   -32766          0
        3   -32766          1
[nemnugin@pd00 ~]$
```

Управление коммуникаторами

Подпрограмма `MPI_Comm_free` помечает коммуникатор `comm` для удаления:

```
int MPI_Comm_free(MPI_Comm *comm)

MPI_Comm_free(comm, ierr)
```

Обмены, связанные с этим коммуникатором, завершаются обычным образом, а сам коммуникатор удаляется только после того, как на него не будет активных ссылок. Данная операция может применяться к коммуникаторам обоих видов (интра- и интер-). `MPI_Comm_free` – *деструктор коммуникатора*.

Управление коммуникаторами

Сравнение двух коммуникаторов (`comm1`) и (`comm2`) выполняется подпрограммой `MPI_Comm_compare`:

```
int MPI_Comm_compare(MPI_Comm comm1, MPI_Comm comm2, int
*result)

MPI_Comm_compare(comm1, comm2, result, ierr)
```

Результат ее выполнения `result` —целое значение, которое равно:

- `MPI_IDENT`, если контексты и группы коммуникаторов совпадают;
- `MPI_CONGRUENT`, если совпадают только группы;
- `MPI_UNEQUAL`, если не совпадают ни группы, ни контексты.

В качестве аргумента нельзя использовать пустой коммуникатор `MPI_COMM_NULL`.

Управление коммуникаторами

К числу операций управления коммуникаторами можно отнести операции `MPI_Comm_size` и `MPI_Comm_rank`. Они позволяют, в частности, распределить роли между процессами в модели «хозяин — работник» (master-slave).

Управление коммуникаторами

С помощью подпрограммы `MPI_Comm_set_name` можно присвоить коммуникатору `comm` строковое имя `name`:

```
int MPI_Comm_set_name(MPI_Comm com, char *name)

MPI_Comm_set_name(com, name, ierr)
```

и наоборот, подпрограмма `MPI_Comm_get_name` возвращает `name` — строковое имя коммуникатора `comm`:

```
int MPI_Comm_get_name(MPI_Comm comm, char *name, int *reslen)

MPI_Comm_get_name(comm, name, reslen, ierr)
```

Имя представляет собой массив символьных значений, длина которого должна быть не менее `MPI_MAX_NAME_STRING`. Длина имени —выходной параметр `reslen`.

Управление коммуникаторами

Две области взаимодействия можно объединить в одну. Подпрограмма `MPI_Intercomm_merge` создает интракоммуникатор `newcomm` из интеркоммуникатора `oldcomm`:

```
int MPI_Intercomm_merge(MPI_Comm oldcomm, int high, MPI_Comm
*newcomm)

MPI_Intercomm_merge(oldcomm, high, newcomm, ierr)
```

Параметр `high` здесь используется для упорядочения групп обоих интракоммуникаторов в `comm` при создании нового коммуникатора.

Управление коммуникаторами

Доступ к удаленной группе, связанной с интеркоммуникатором `comm`, можно получить, обратившись к подпрограмме:

```
int MPI_Comm_remote_group(MPI_Comm comm, MPI_Group *group)

MPI_Comm_remote_group(comm, group, ierr)
```

Ее выходным параметром является удаленная группа `group`.

Управление коммуникаторами

Подпрограмма `MPI_Comm_remote_size` определяет размер удаленной группы, связанной с интеркоммуникатором `comm`:

```
int MPI_Comm_remote_size(MPI_Comm comm, int *size)

MPI_Comm_remote_size(comm, size, ierr)
```

Ее выходной параметр `size` —количество процессов в области взаимодействия, связанной с коммуникатором `comm`.

Операции обмена между группами процессов

Интеробмены

При выполнении интеробмена процессу-источнику сообщения указывается ранг адресата относительно удаленной группы, а процессу-получателю — ранг источника (также относительно удаленной по отношению к получателю группы). Обмен выполняется между лидерами обеих групп (MPI-1). Предполагается, что в обеих группах есть, по крайней мере, по одному процессу, который может обмениваться сообщениями со своим партнером.

Интеробмены

Интеробмен возможен, только если создан соответствующий интеркоммуникатор, а это можно сделать с помощью подпрограммы:

```
int MPI_Intercomm_create(MPI_Comm local_comm, int local_leader,
MPI_Comm peer_comm, int remote_leader, int tag, MPI_Comm
*new_intercomm)

MPI_Intercomm_create(local_comm, local_leader, peer_comm,
remote_leader, tag, new_intercomm, ierr)
```

Входные параметры этой подпрограммы:

- `local_comm` —локальный интракоммуникатор;
- `local_leader` —ранг лидера в локальном коммуникаторе (обычно 0);
- `peer_comm` —удаленный коммуникатор;
- `remote_leader` —ранг лидера в удаленном коммуникаторе (обычно 0);
- `tag` —тег интеркоммуникатора, используемый лидерами обеих групп для обменов в контексте родительского коммуникатора.

Интеробмены

Выходной параметр —интеркоммуникатор (`new_intercomm`). «Джокеры» в качестве параметров использовать нельзя. Вызов этой подпрограммы должен выполняться в обеих группах процессов, которые должны быть связаны между собой. В каждом из этих вызовов используется локальный интракоммуникатор, соответствующий данной группе процессов.

ВНИМАНИЕ

При работе с `MPI_Intercomm_create` локальная и удаленная группы процессов не должны пересекаться, иначе возможны «тупики».

Интеробмены

Пример создания интеркоммуникаторов

```
#include "mpi.h"
#include <stdio.h>
int main(int argc,char *argv[])
{
 int counter, message, myid, numprocs, server;
 int color, remote_leader_rank, i, ICTAG = 0;
 MPI_Status status;
 MPI_Comm oldcommdup, splitcomm, oldcomm, inter_comm;
 MPI_Init(&argc, &argv);
 oldcomm = MPI_COMM_WORLD;
 MPI_Comm_dup(oldcomm, &oldcommdup);
 MPI_Comm_size(oldcommdup, &numprocs);
 MPI_Comm_rank(oldcommdup, &myid);
 server = numprocs – 1;
 color = (myid == server);
 MPI_Comm_split(oldcomm, color, myid, &splitcomm);
```

Интеробмены

```
if(!color) {
 remote_leader_rank = server;
 }
 else {
 remote_leader_rank = 0;
 }
 MPI_Intercomm_create(splitcomm, 0, oldcommdup,
remote_leader_rank, ICTAG, &inter_comm);
 MPI_Comm_free(&oldcommdup);
 if (myid == server) {
 for(i = 0; i<server; i++){
  MPI_Recv(&message, 1, MPI_INT, i, MPI_ANY_TAG, inter_comm,
&status);
  printf("Process rank %i received %i from %i\n", myid,
message, status.MPI_SOURCE);}
}
```

Интеробмены

```
else{
 counter = myid;
 MPI_Send(&counter, 1, MPI_INT,  0, 0, inter_comm);
 printf("Process rank %i send %i\n", myid, counter);
}
 MPI_Comm_free(&inter_comm );
 MPI_Finalize();
}
```

Интеробмены

В этой программе процессы делятся на две группы: первая состоит из одного процесса (процесс с максимальным рангом в исходном коммуникаторе `MPI_COMM_WORLD`), это —«сервер», а во вторую входят все остальные процессы. Между этими группами и создается интеркоммуникатор `inter_comm`. Процессы-клиенты передают серверу сообщения. Сервер принимает эти сообщения с помощью подпрограммы стандартного блокирующего двухточечного приема и выводит их на экран. «Ненужные» коммуникаторы удаляются. Распечатка вывода этой программы выглядит следующим образом:

Интеробмены

```
[nemnugin@pd00 ~]$ mpiexec -n 10 ./a.out
Process rank 0 send 0
Process rank 1 send 1
Process rank 8 send 8
Process rank 5 send 5
Process rank 3 send 3
Process rank 7 send 7
Process rank 9 received 0 from 0
Process rank 9 received 1 from 1
Process rank 4 send 4
Process rank 2 send 2
Process rank 6 send 6
Process rank 9 received 2 from 2
Process rank 9 received 3 from 3
Process rank 9 received 4 from 4
Process rank 9 received 5 from 5
Process rank 9 received 6 from 6
Process rank 9 received 7 from 7
Process rank 9 received 8 from 8
[nemnugin@pd00 ~]$
```

Заключение

В этой лекции мы рассмотрели:

- способы создания групп процессов;
- способы создания коммуникаторов;
- особенности использования интра- и интеркоммуникаторов;
- организацию обменов между двумя группами процессов.

Задания для самостоятельной работы

Решения следует высылать по электронной почте:

parallel-g112@yandex.ru

Задания для самостоятельной работы

1. Напишите параллельную программу, в которой создаются *N* групп процессов, и обмен между этими группами выполняется по замкнутому кольцу.
2. Напишите программу, в которой все процессы наделяются логической топологией двумерной решётки. Эта решётка разбивается на блоки, каждому из которых сопоставляется собственный коммуникатор.

Тема следующей лекции

Пользовательские типы. Виртуальные топологии