

Технология программирования MPI (5)

Антонов Александр Сергеевич,
к.ф.-м.н., вед.н.с. лаборатории
Параллельных информационных технологий
НИВЦ МГУ

Летняя суперкомпьютерная академия
Москва, 2018

МРІ

Группы и коммунікаторы

MPI

Необходимость групп и коммуникаторов:

- дать возможность некоторой группе процессов работать над своей независимой подзадачей;
- если только часть процессов должна обмениваться данными, бывает удобно завести для их взаимодействия отдельный коммуникатор;
- при создании библиотек подпрограмм нужно гарантировать, что пересылки данных в библиотечных модулях не пересекутся с пересылками в основной программе.

МРІ

**Функции для работы с
группами процессов**

MPI

Группа – упорядоченное множество процессов. Каждому процессу в группе сопоставлено целое число – номер (ранг).

MPI_GROUP_EMPTY – пустая группа.

MPI_GROUP_NULL – значение, используемое для ошибочной группы.

Новую группу можно создать только из уже существующих групп или коммутаторов.

MPI

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

Получение группы **group**, соответствующей коммуникатору **comm**.

Поскольку изначально существует единственный нетривиальный коммуникатор **MPI_COMM_WORLD**, обычно в начале программы нужно получить соответствующую ему группу процессов:

```
MPI_Comm_group(MPI_COMM_WORLD, &group);
```

MPI

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

Создание группы **newgroup** из пересечения групп **group1** и **group2**. Полученная группа содержит все процессы группы **group1**, входящие также в группу **group2** и упорядоченные так же, как в первой группе.

MPI

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

Создание группы **newgroup** из объединения групп **group1** и **group2**. Полученная группа содержит все процессы группы **group1** в прежнем порядке, за которыми следуют процессы группы **group2**, не вошедшие в группу **group1**, также в прежнем порядке.

MPI

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

Создание группы **newgroup** из разности групп **group1** и **group2**. Полученная группа содержит все элементы группы **group1**, не входящие в группу **group2** и упорядоченные, как в первой группе.

MPI

Пусть в группу **gr1** входят процессы **0, 1, 2, 4, 5**, а в группу **gr2** – процессы **0, 2, 3**.

```
MPI_Group_intersection(gr1, gr2, &newgr1);
```

```
MPI_Group_union(gr1, gr2, &newgr2);
```

```
MPI_Group_difference(gr1, gr2, &newgr3);
```

После ЭТИХ ВЫЗОВОВ:

newgr1: 0, 2;

newgr2: 0, 1, 2, 4, 5, 3;

newgr3: 1, 4, 5.

MPI

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

Создание группы **newgroup** из **n** процессов группы **group** с рангами **ranks[0]**, ..., **ranks[n-1]**, причём рангу **ranks[i]** в старой группе соответствует ранг **i** в новой группе. При **n=0** создается пустая группа **MPI_GROUP_EMPTY**.

MPI

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

Создание группы **newgroup** из процессов группы **group**, исключая процессы с рангами **ranks[0], ..., ranks[n-1]**, причём порядок процессов в новой группе соответствует порядку процессов в старой группе. При **n=0** создаётся группа, идентичная старой группе.

MPI

```
MPI_Comm_group(MPI_COMM_WORLD, &group);  
size1 = size/2;  
for(i=0; i<size1; i++) ranks[i] = i;  
MPI_Group_incl(group, size1, ranks, &group1);  
MPI_Group_excl(group, size1, ranks, &group2);
```

MPI

```
int  
MPI_Group_range_incl(MPI_Group  
group, int n, int ranges[][3],  
MPI_Group *newgroup)
```

Массив **ranges** состоит из **n** троек типа **(first₁, last₁, stride₁), ..., (first_n, last_n, stride_n)**.

MPI

Создаваемая группа **newgroup** будет содержать процессы с номерами

$first_1, first_1+stride_1, \dots, first_1+k_1*stride_1, \dots,$

$first_n, first_n+stride_n, \dots, first_n+k_n*stride_n.$

Все вычисляемые таким образом номера процессов должны быть различны, и процессы с такими номерами должны

MPI

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

Данная процедура создает новую группу **newgroup**, исключая из группы **group** процессы, номера которых определяются, как описано для процедуры **MPI_Group_range_incl**.

MPI

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

Определение количества **size** процессов в группе **group**.

MPI

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

Определение номера **rank** процесса в группе **group**. Если процесс не входит в группу **group**, то возвращается значение **MPI_UNDEFINED**.

MPI

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

В массиве **ranks2** возвращаются ранги в группе **group2** процессов с рангами **ranks1** в группе **group1**. Если процесс из группы **group1** не входит в группу **group2**, то для него возвращается значение **MPI_UNDEFINED**.

MPI

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

Сравнение групп **group1** и **group2**. Если совпадают, то возвращается значение **MPI_IDENT**. Если отличаются только рангами процессов, то возвращается значение **MPI_SIMILAR**. Иначе возвращается значение **MPI_UNEQL**.

MPI

```
int MPI_Group_free(MPI_Group  
*group)
```

Уничтожение группы **group**. Переменная **group** принимает значение **MPI_GROUP_NULL**.

MPI

```
#include <stdio.h>
#include "mpi.h"
int main(int argc, char **argv)
{
    int rank, i, size, size1, n;
    MPI_Status status;
    MPI_Group group, group1, group2;
    int ranks[128], rank1, rank2, rank3;
    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_group(MPI_COMM_WORLD, &group);
```

MPI

```
size1 = size/2;
for(i = 0; i<size1; i++) ranks[i] = i;
MPI_Group_incl(group, size1, ranks, &group1);
MPI_Group_excl(group, size1, ranks, &group2);
MPI_Group_rank(group1, &rank1);
MPI_Group_rank(group2, &rank2);
if(rank1==MPI_UNDEFINED){
    if(!(size%2) || rank!=size-1)
MPI_Group_translate_ranks(group1, 1, &rank2,
group, &rank3);
    else rank3 = MPI_PROC_NULL;
}
```

MPI

```
    else MPI_Group_translate_ranks(group2, 1,
&rank1, group, &rank3);

    MPI_Sendrecv(&rank, 1, MPI_INT, rank3, 1, &n,
1, MPI_INT, rank3, 1, MPI_COMM_WORLD, &status);
    MPI_Group_free(&group);
    MPI_Group_free(&group1);
    MPI_Group_free(&group2);
    printf("process %d n=%d\n", rank, n);
    MPI_Finalize();
}
```


МРІ

Функции для работы с коммуникаторами

MPI

Коммуникатор – контекст обмена группы. В операциях обмена используются только коммуникаторы!

MPI_COMM_WORLD – коммуникатор для всех процессов приложения.

MPI_COMM_NULL – значение, используемое для ошибочного коммуникатора.

MPI_COMM_SELF – коммуникатор, включающий только вызвавший процесс.

MPI

Коммуникатор даёт возможность независимых обменов. Каждой группе процессов может соответствовать несколько коммуникаторов, но каждый коммуникатор соответствует только одной группе.

Создание коммуникатора является коллективной операцией и требует операции межпроцессного обмена, поэтому такие процедуры могут оказаться весьма затратными по времени.

MPI

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

Создание нового коммуникатора **newcomm** с той же группой процессов и атрибутами, что и у коммуникатора **comm**.

MPI

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

Процедура создает новый коммуникатор **newcomm** из коммуникатора **comm** для группы процессов **group**, которая должна являться подмножеством группы, связанной с коммуникатором **comm**.

MPI

Разные процессы могут указывать разный аргумент **group**, некоторые могут указывать значение **MPI_GROUP_EMPTY**. Но если некоторый процесс указал нетривиальную группу, то у всех процессов этой группы параметр **group** должен быть одинаковым. Так можно создать набор непересекающихся коммуникаторов. Вызов должен встретиться во всех процессах коммуникатора **comm**. На процессах, не принадлежащих ни одной из **group**, вернётся **MPI_COMM_NULL**.

MPI

```
#include <stdio.h>
#include "mpi.h"
int main(int argc, char **argv)
{
    int rank, size, i, rbuf;
    MPI_Group group, new_group;
    MPI_Comm new_comm;
    int ranks[128], new_rank;
    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_group(MPI_COMM_WORLD, &group);
```

MPI

```
for(i=0; i<size/2; i++) ranks[i] = i;
if(rank < size/2) MPI_Group_incl(group,
size/2, ranks, &new_group);
else MPI_Group_excl(group, size/2, ranks,
&new_group);
MPI_Comm_create(MPI_COMM_WORLD, new_group,
&new_comm);
MPI_Allreduce(&rank, &rbuf, 1, MPI_INT,
MPI_SUM, new_comm);
MPI_Group_rank(new_group, &new_rank);
printf("rank=%d newrank=%d rbuf=%d\n", rank,
new_rank, rbuf);
MPI_Finalize();
}
```


MPI

```
int MPI_Comm_split(MPI_Comm  
comm, int color, int key,  
MPI_Comm *newcomm)
```

Разбиение коммуникатора **comm** на несколько по числу значений параметра **color**. В одну подгруппу попадают процессы с одним значением **color**. Процессы с бóльшим значением **key** получают больший ранг.

MPI

Процессы, которые не должны войти в новые группы, указывают в качестве **color** константу **MPI_UNDEFINED**. Им в параметре **newcomm** вернется значение **MPI_COMM_NULL**.

```
MPI_Comm_split(MPI_COMM_WORLD, rank%3, rank,  
new_comm)
```

MPI

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

Сравнение **comm1** и **comm2**. Если совпадают, то в **result** возвращается **MPI_IDENT**.

Если соответствуют одинаковым группам, а отличаются контекстом, то

MPI_CONGRUENT. Если соответствуют группам с одними процессами, но разной нумерацией, то возвращается

MPI_SIMILAR. Иначе - **MPI_UNEQUAL**.

MPI

```
int MPI_Comm_free(MPI_Comm comm)
```

Удаление коммуникатора **comm**. Переменной **comm** присваивается значение **MPI_COMM_NULL**.

MPI

```
#include <stdio.h>
#include "mpi.h"
int main(int argc, char **argv)
{
    int rank, size, rank1;
    MPI_Comm comm_revs;
    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_split(MPI_COMM_WORLD, 1, size-rank,
&comm_revs);
    MPI_Comm_rank(comm_revs, &rank1);
    printf("rank = %d rank1 = %d\n", rank, rank1);
    MPI_Comm_free(&comm_revs);
    MPI_Finalize();
}
```

MPI

Задание 8: Реализуйте на MPI разбиение процессов приложения на две группы, в одной из которых осуществляется обмен по кольцевой топологии, а в другой – коммуникации по схеме «мастер – рабочие» (с использованием любых изученных функций).