Группы и коммуникаторы

НОЦ МФТИ 2011

Введение

- Уже в какой-то мере знакомы с коммуникаторами по MPI_COMM_WORLD
- Данный коммуникатор по умолчанию включает все запущенные процессы и позволяет вести обмен информацией как между двумя процессами (p-t-p) так и коллективные пересылки
- В ряде случаев возникает необходимость коллективных взаимодействий между группой процессов
- Сегодня мы рассмотрим методы для создания коммуникаторов в MPI

Коммуникаторы

- Два вида коммуникаторов:
  - Intra-коммуникатор
  - Inter-коммуникатор
- Intra-коммуникатор имеет место при обмене сообщениями между процессами
- Inter-коммуникатор работает с обменом между intra-коммуникаторами
- Любой intra-коммуникатор состоит из части процессов MPI_COMM_WORLD
- В основном коммуникаторы используются для создания виртуальных топологий, что позволяет улучшить архитектуру программ, их поддержку и читаемость кода

Функции в MPI

- Для работы с коммуникаторами в MPI существует следующий набор функций:
  - MPI_COMM_GROUP
  - MPI_GROUP_INCL
  - MPI_GROUP_EXCL
  - MPI_GROUP_RANK
  - MPI_GROUP_FREE
  - MPI_COMM_CREATE
  - MPI_COMM_SPLIT
- Рассмотрим их подробнее

MPI_Comm_group

- Используется для получения группы процессов в коммуникаторе
- Возвращает структуру типа MPI_Group со списком процессов

int MPI_Comm_group(MPI_Comm comm, MPI_Group *group)

```
#include "mpi.h"
MPI_Comm  comm_world;
MPI_Group group_world;

comm_world = MPI_COMM_WORLD;
MPI_Comm_group(comm_world, &group_world);
```

MPI_Group_incl

int MPI_Group_incl(MPI_Group old_group, int count, int *members, MPI_Group *new_group)

- Предназначена для создания новой группы из процессов принадлежащих старой группе с определенными номерами
- old_group — старая группа, откуда выделяются процессы
- count — число выделяемых процессов
- members — указатель на массив с номерами процессов
- new_group — новая группа

Пример

Выделим четные процессы в отдельную группу:

```
#include "mpi.h"
MPI_Group group_world, odd_group, even_group;
int i, p, Neven, Nodd, members[8], ierr;

MPI_Comm_size(MPI_COMM_WORLD, &p);
MPI_Comm_group(MPI_COMM_WORLD, &group_world);

Neven = (p+1)/2;    /* processes of MPI_COMM_WORLD are divided */
Nodd = p - Neven;   /* into odd- and even-numbered groups */
for (i=0; i<Neven; i++) {   /* "members" determines members of even_group */
    members[i] = 2*i;
};

MPI_Group_incl(group_world, Neven, members, &even_group);
```

MPI_Group_incl

- В предыдущем примере из коммуникатора MPI_COMM_WORLD выделены процессы с четными номерами
- В новом коммуникаторе процессоры опять пронумерованы от 0 до Neven-1, причем порядок нумерации совпадает с порядком в массиве members
- Старый коммуникатор продолжает содержать в себе все процессы MPI_COMM_WORLD

MPI_Group_excl

```
int MPI_Group_excl( MPI_Group group, int count, int
  *nonmembers, MPI_Group *new_group )
```

- Используется для создания новой группы путем удаления процессов из старой
- group — старая группа
- count — число удаляемых процессов
- nonmembers — массив с номерами процессов
- new_group — новая группа

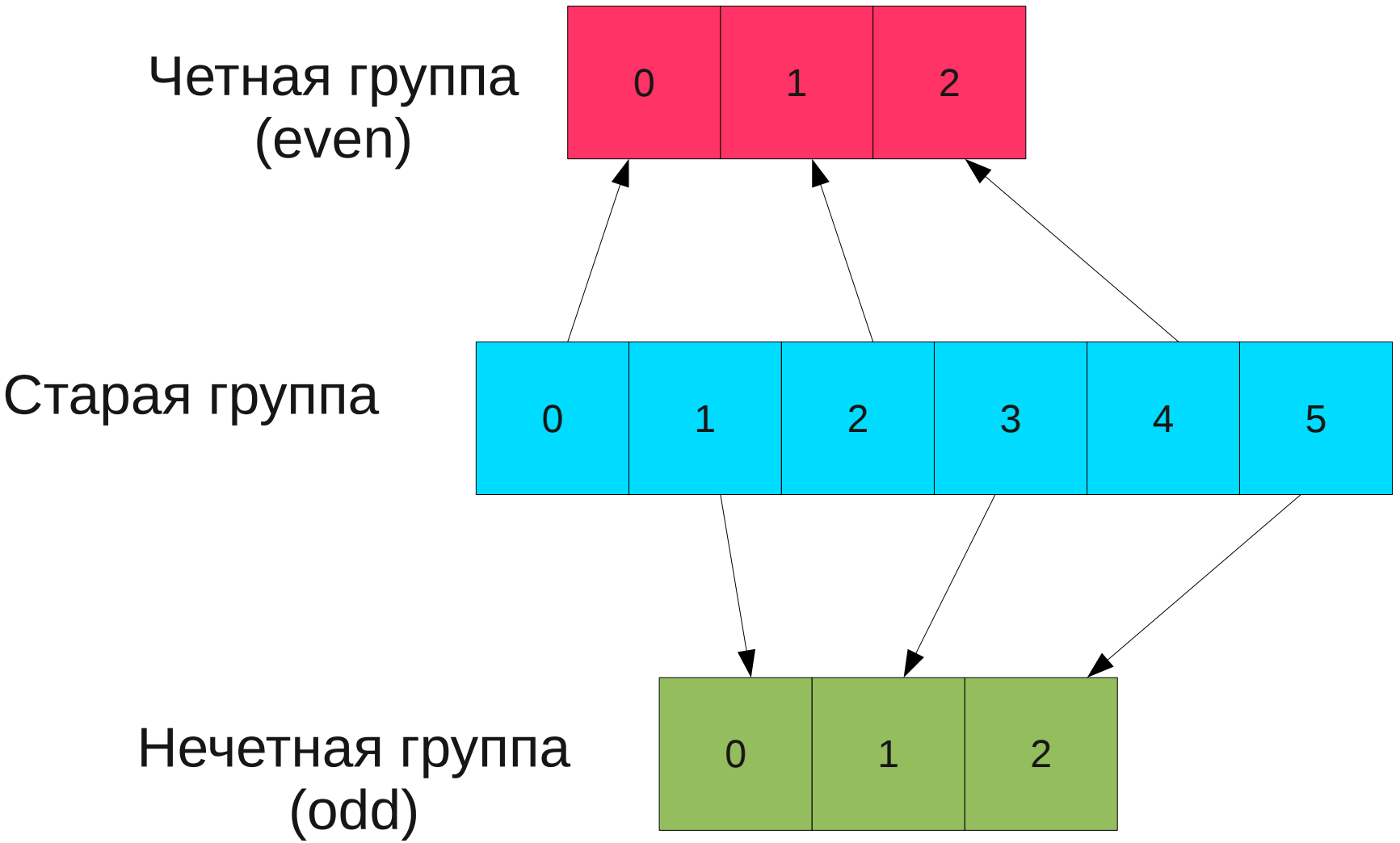
Пример

Выделим нечетные процессы в отдельную группу:

```
#include "mpi.h"
MPI_Group group_world, odd_group, even_group;
int i, p, Neven, Nodd, nonmembers[8], ierr;

MPI_Comm_size(MPI_COMM_WORLD, &p);
MPI_Comm_group(MPI_COMM_WORLD, &group_world);

Neven = (p+1)/2;    /* processes of MPI_COMM_WORLD are divided */
Nodd = p - Neven;   /* into odd- and even-numbered groups */
for (i=0; i<Neven; i++) {   /* "nonmembers" are even-numbered procs */
    nonmembers[i] = 2*i;
};

MPI_Group_excl(group_world, Neven, nonmembers, &odd_group);
```

MPI_group_excl

- В данном примере создана новая группа, содержащая нечетные процессы коммуникатора MPI_COMM_WORLD
- В отличие от MPI_Group_incl, данная функция принимает список процессов, которые необходимо исключить
- Процессы пронумерованы от 0 до Nodd-1, причем порядок номеров в старой и новой группе совпадает


Четная группа
(even)
0
1
2
Старая группа
0
1
2
3
4
5
Нечетная группа
(odd)
0
1
2

MPI_Group_rank

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

- Используется для получения номера процесса в группе
- Аналог MPI_Comm_rank
- В случае отсутствия процесса в группе возвращает MPI_UNDEFINED

Пример

```
#include "mpi.h"
MPI_Group group_world, worker_group;
int i, p, ierr, group_rank, re[1] = {0};

MPI_Comm_size(MPI_COMM_WORLD, &p);
MPI_Comm_group(MPI_COMM_WORLD, &group_world);
MPI_Group_excl(group_world, 1, re, &worker_group);

MPI_Group_rank(worker_group, &group_rank);
```

MPI_Group_free

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

- Используется для освобождения группы, когда она больше не нужна
- Не очищает коммуникатор, к которому принадлежит
- Для очистки коммуникатора существует процедура MPI_Comm_free

MPI_Comm_create

int MPI_Comm_create(MPI_Comm old_comm, MPI_Group group, MPI_Comm *new_comm)

- Используется для создания нового коммуникатора
- Коммуникатор создается на основе группы и старого коммуникатора
- Группа должна быть создана на основе группы старого коммуникатора

Пример

```
#include "mpi.h"
MPI_Comm  comm_world, comm_worker;
MPI_Group group_world, group_worker;
int ierr;

comm_world = MPI_COMM_WORLD;
MPI_Comm_group(comm_world, &group_world);
MPI_Group_excl(group_world, 1, 0, &group_worker);  /* process 0 not member */

MPI_Comm_create(comm_world, group_worker, &comm_worker);
```

Особенности

- Коллективная операция и должна быть вызвана на всех процессах из старого коммуникатора
- Для процессов вне группы возвращает MPI_COMM_NULL
- После создание новые коммуникаторы могут быть освобождены используя MPI_Comm_free

MPI_Comm_split

- Используется для создания новых коммуникаторов
- Позволяет создавать новые коммуникаторы из старого по некоторым заданным свойствам
- Например при матричных вычислениях можно выделить наборы отдельных колонок в коммуникаторы

MPI_Comm_split

```
int MPI_Comm_split( MPI_Comm old_comm, int color, int key, MPI_Comm
    *new_comm )
```

- old_comm — старый коммуникатор
- color — разделяет процессы с одним цветом в одни группы
- key - в каждой группе определяет порядок присвоения номера
- new_comm — новый коммуникатор

Пример

```
/* logical 2D topology with nrow rows and mcol columns */
irow = rank/mcol;        /* logical row number */
jcol = mod(rank, mcol);  /* logical column number */
comm2D = MPI_COMM_WORLD;

MPI_Comm_split(comm2D, irow, jcol, row_comm);
MPI_Comm_split(comm2D, jcol, irow, col_comm);
```

Пример

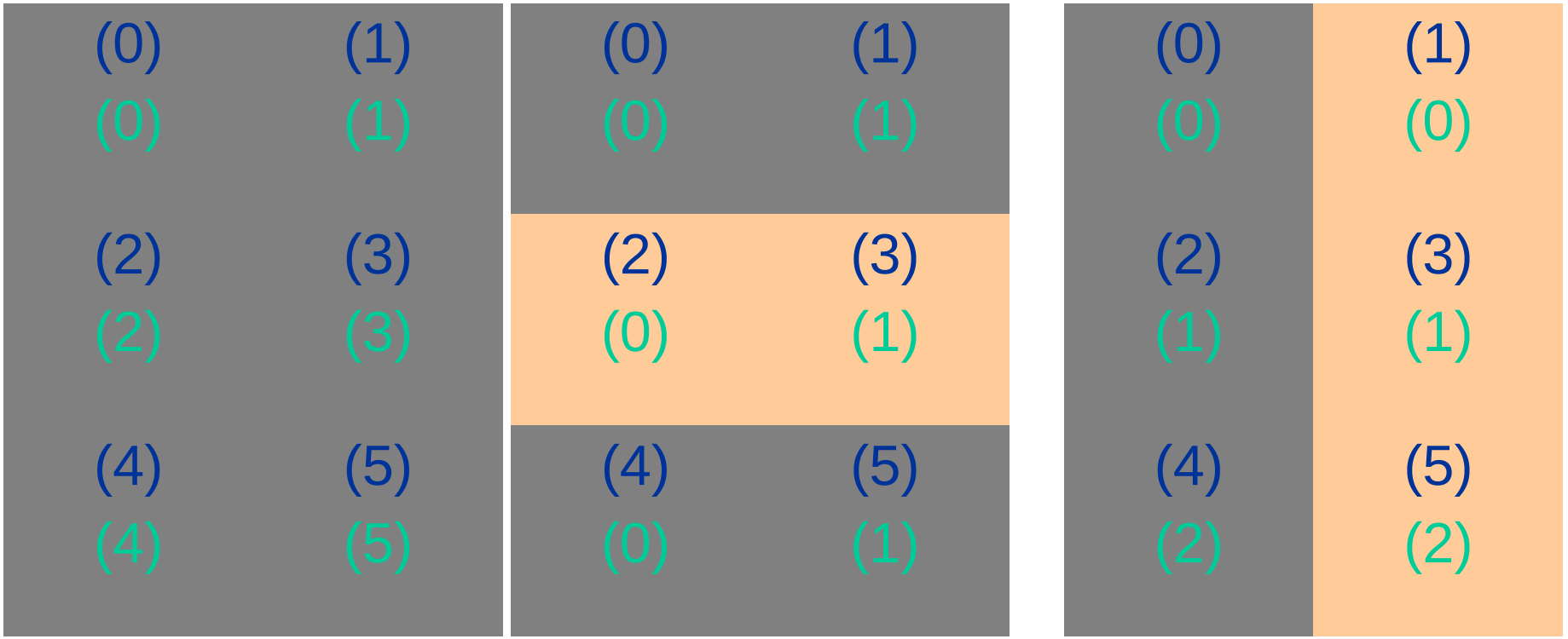
- Создает двумерную топологию по строкам и столбцам
- Для примера возьмет 6 процессов: 0...5
- Эти процессы могут быть организованы в двумерную сеть размером 3 x 2
- В зависимости от того, рассматриваем строку или столбец у процесса может быть разный номер

Процесс разделения

- Вначале irow используется как цвет, а jcol как ключ для нумерации процессов
- Далее jcol используется как цвет, а irow для нумерации, что дает другую топологию

rank	0	1	2	3	4	5
irow	0	0	1	1	2	2
jcol	0	1	0	1	0	1

MPI_Comm_split



MPI_Comm_split

Логическая сеть

Построчная

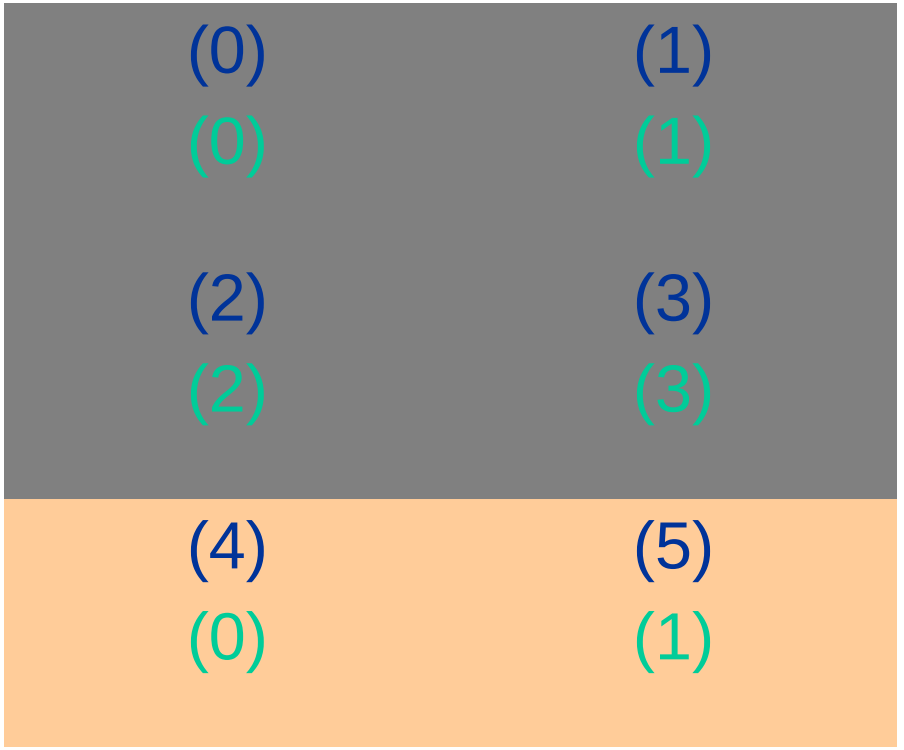
Поколоночная

Пример

Еще один пример, теперь разделяем колонки еще на 2 группы, первые две строки в одну группу и третью в отдельную

```
/* simple example of 6 processes divided into 2 groups; */
/* 1st 4 belongs to group 0 and remaining two to group 1 */
group = rank/4;                         /* group0:0,1,2,3; group1:4,5 */
index = rank - row_group*4;             /* group0:0,1,2,3; group1:0,1 */
err = MPI_Comm_split(comm2D, group, index, row_comm);
```

Результат



Результат работы

Пример разбиения на четные и нечетные

```
#include "stdio.h"
#include "mpi.h"
void main(int argc, char *argv[])
{
    int Iam, p;
    int Neven, Nodd, members[6], even_rank, odd_rank;
    MPI_Group group_world, even_group, odd_group;

    /* Starts MPI processes ... */
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &Iam);            /* get current process
    id */
    MPI_Comm_size(MPI_COMM_WORLD, &p);              /* get number of
    processes */
    Neven = (p + 1)/2;      /* All processes of MPI_COMM_WORLD are divided */
    Nodd = p - Neven;       /* into 2 groups, odd- and even-numbered groups */
    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);
```

Продолжение

```
MPI_Barrier(MPI_COMM_WORLD);
    if(Iam == 0) {
            printf("MPI_Group_incl/excl Usage Example\n");
            printf("\n");
            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("MPI_UNDEFINED is set to     %d\n",
    MPI_UNDEFINED);
            printf("\n");
            printf("     Iam     even      odd\n");
    }
    MPI_Barrier(MPI_COMM_WORLD);
    MPI_Group_rank(even_group, &even_rank);
    MPI_Group_rank( odd_group,  &odd_rank);
    printf("%8d %8d %8d\n",Iam, even_rank, odd_rank);
    MPI_Finalize();                                 /* let MPI finish
    up ...  */
}
```

Вывод

```
MPI_Group_incl/excl Usage Example

Number of processes is 6
Number of odd processes is 3
Number of even processes is 3
members[0] is assigned rank 2
members[1] is assigned rank 0
members[2] is assigned rank 4

MPI_UNDEFINED is set to -32766

    Iam     even      odd
      0        1   -32766
      1   -32766        0
      4        2   -32766
      2        0   -32766
      5   -32766        2
      3   -32766        1
```

Отрицательные значения означают, что процесс не входит в группу

Вопросы.