Библиотека MPI: Дополнительные функции

Лекция 5

Функции для работы с производными типами данных

Производные типы данных

- регистрируются вызовом функции MPI_Type_commit
- ненужные типы уничтожаются функцией MPI_Type_free
- Предопределенные типы MPI считаются зарегистрированными

Характеристики типов

- Протяженность типа : адрес последней ячейки данных - адрес первой ячейки данных + длина последней ячейки данных

  int MPI_Type_extent(MPI_Datatype datatype, MPI_Aint *extent)

- Размер типа: сумме длин всех базовых элементов определяемого типа

  int MPI_Type_size(MPI_Datatype datatype, int *size)

Основные функции

```
int MPI_Type_contiguous(int count, MPI_Datatype oldtype,
      MPI_Datatype *newtype)
```

oldtype=MPI_REAL

count = 4

newtype

Рис. 12. Схема построения

Основные функции

int MPI_Type_vector(int count, int blocklength, int stride,
MPI_Datatype oldtype, MPI_Datatype *newtype)

oldtype=MPI_REAL 

count = 3, blocklenght=2, stride=3

newtype

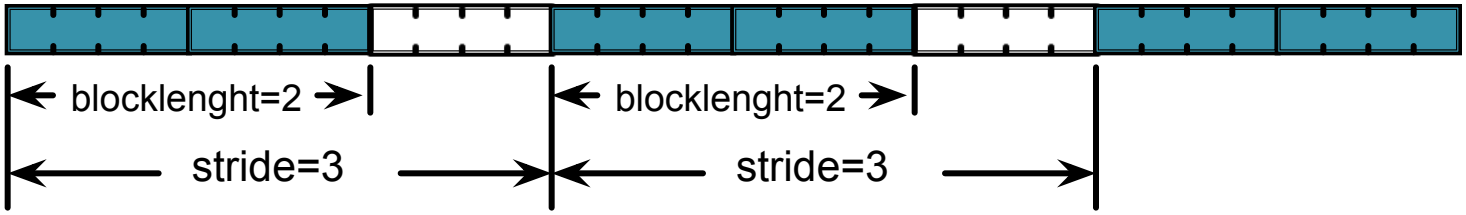


Рис. 13. Схема построения

Основные функции

int MPI_Type_hvector(int count, int blocklength, MPI_Aint stride,
MPI_Datatype oldtype, MPI_Datatype *newtype)

oldtype=MPI_REAL 

count = 3, blocklenght=2, stride=10

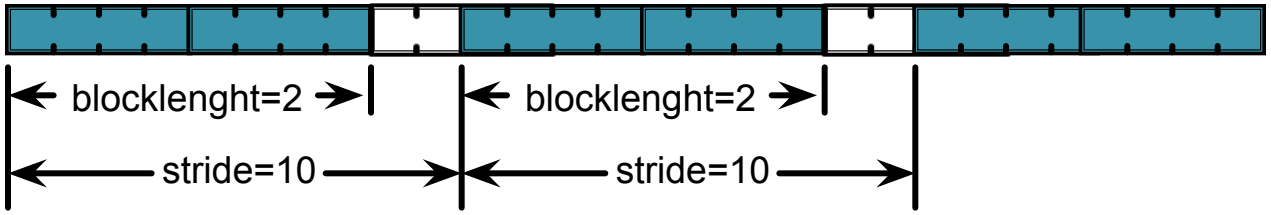
newtype 

Рис. 14. Схема построения

Основные функции

```
int MPI_Type_indexed(int count, int *array_of_blocklengths,
        int *array_of_displacements, MPI_Datatype oldtype,
        MPI_Datatype *newtype)
```

oldtype=MPI_REAL 

count = 2, blocklenght=(2,4), displacement=(0,4)

newtype

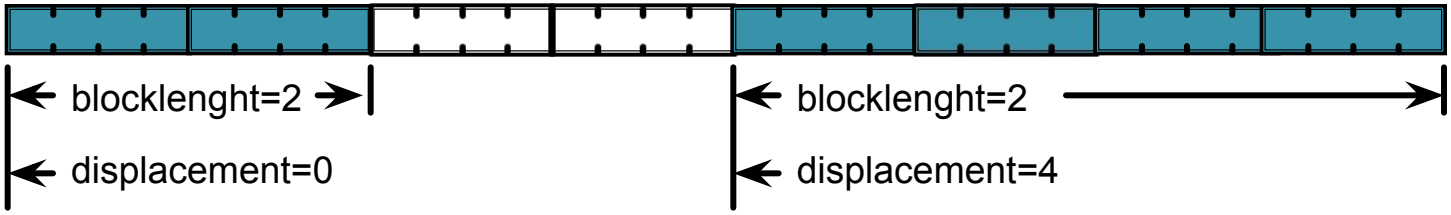


Рис. 15. Схема построения

Основные функции

int MPI_Type_hindexed(int count, int *array_of_blocklengths,
MPI_Aint *array_of_displacements, MPI_Datatype
oldtype, MPI_Datatype *newtype)

oldtype=MPI_REAL 

count = 3, blocklenght=(2,3,1), displacement=(0,10,26)

newtype

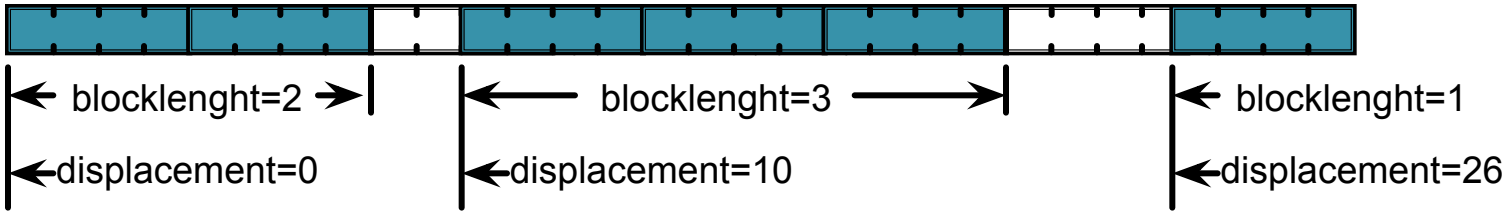


Рис. 16. Схема построения

Основные функции

```
int MPI_Type_struct(int count, int *array_of_blocklengths,
                    MPI_Aint *array_of_displacements, MPI_Datatype
                    *array_of_types, MPI_Datatype *newtype)
```

oldtype=(MPI_INT, MPI_SHORT, MPI_CHAR)

count = 3, blocklenght=(1,6,4), displacement=(0,12,26)

newtype

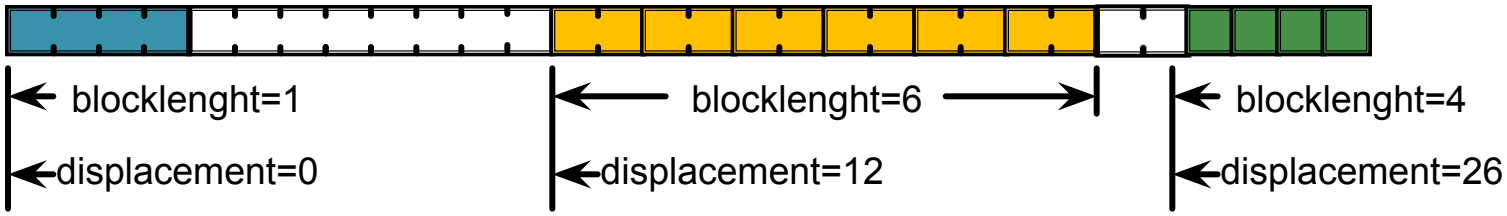


Рис. 17. Схема построения

Основные функции

int MPI_Type_commit(MPI_Datatype *datatype)

int MPI_Type_free(MPI_Datatype *datatype)

Функция MPI_Type_free устанавливает описатель типа в состояние MPI_DATATYPE_NULL

Передача упакованных данных

```
int MPI_Pack(void* inbuf, int incount, MPI_Datatype
    datatype, void *outbuf,
    int outsize, int *position, MPI_Comm comm)
int MPI_Pack_size(int incount, MPI_Datatype
    datatype, MPI_Comm comm, int *size)

int MPI_Unpack(void* inbuf, int insize,
    int *position, void *outbuf, int outcount,
    MPI_Datatype datatype, MPI_Comm comm)
```

/* Упаковка данных */

```
if (myrank == 0) {
        position = 0;
        MPI_Pack(&x, 1, MPI_DOUBLE, buff, 100,
        &position, MPI_COMM_WORLD);
        MPI_Pack(&y, 1, MPI_DOUBLE, buff, 100,
        &position, MPI_COMM_WORLD);
        MPI_Pack(a, 2, MPI_INT, buff, 100,
        &position, MPI_COMM_WORLD);
}
```

/* Рассылка упакованного сообщения */

```
MPI_Bcast(buff, position, MPI_PACKED, 0,
        MPI_COMM_WORLD);
```

/* Распаковка сообщения во всех процессах */

```
if (myrank != 0) {
   position = 0;
   MPI_Unpack(buff, 100, &position, &x, 1,
   MPI_DOUBLE, MPI_COMM_WORLD);
   MPI_Unpack(buff, 100, &position, &y, 1,
   MPI_DOUBLE, MPI_COMM_WORLD);
   MPI_Unpack(buff, 100, &position, a, 2,
   MPI_INT, MPI_COMM_WORLD);
}
```

Функции для работы с группами и коммуникаторами

Основные понятия

Два взаимосвязанных механизма:

- функции для работы с группами процессов как упорядоченными множествами
- функции для работы с коммуникаторами, для создания новых коммуникаторов как описателей новых областей связи

Основные понятия (группа)

- **Группа** – упорядоченное множество процессов
- Специальный тип данных MPI_Group
- Две предопределенные группы:
  - **MPI_GROUP_EMPTY**
  - **MPI_GROUP_NULL**
- Нет группы, соответствующей коммуникатору MPI_COMM_WORLD

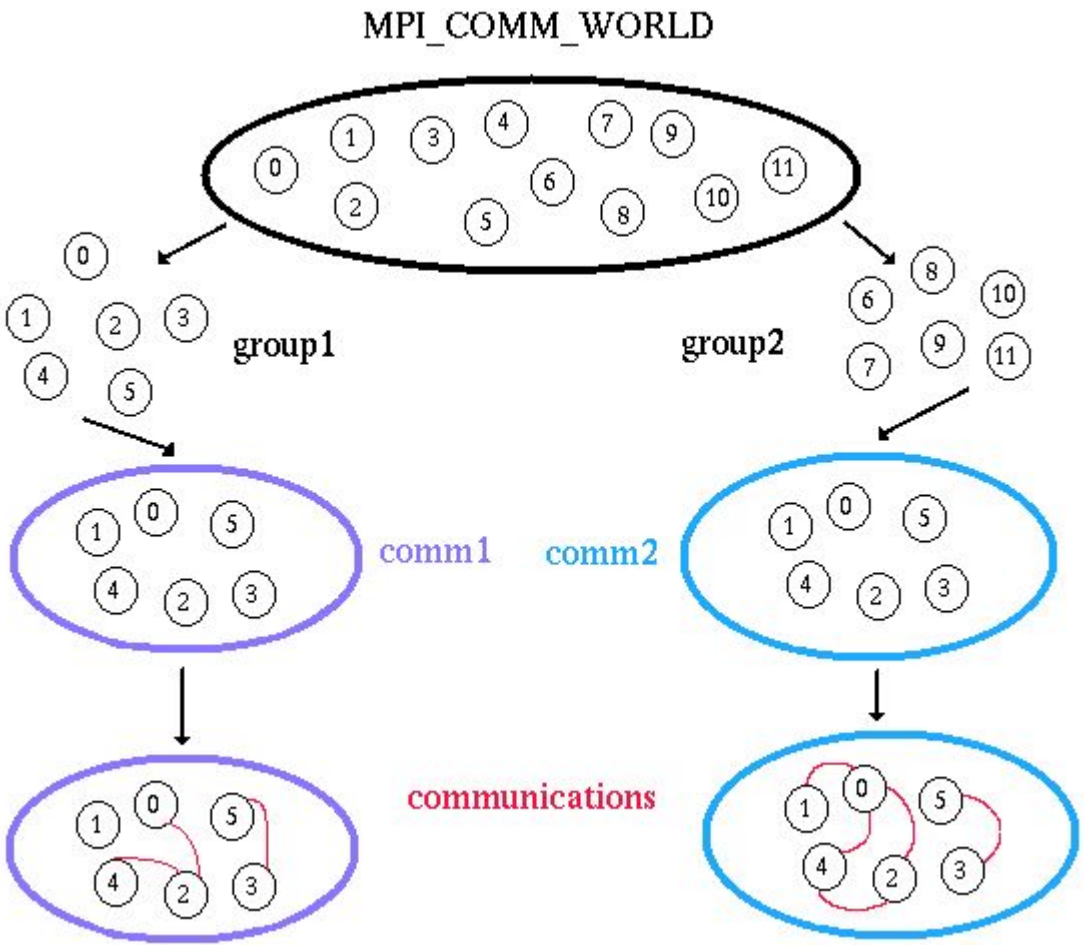
Основные понятия (коммуникатор)

- **Коммуникатор** – скрытый объект с некоторым набором атрибутов, правилами его создания, использования и уничтожения
- Коммуникатор описывает некоторую область связи
- Два предопределенных коммуникатора:
  - **MPI_COMM_WORLD**
  - **MPI_COMM_SELF**

Алгоритм работы

1. **MPI_Comm_group**: Получаем описатель глобальной группы, содержащей все процессы из MPI_COMM_WORLD
2. **MPI_Group_incl**: Формируем новую группу как подмножество глобальной группы
3. **MPI_Comm_create**: Создаем новый коммуникатор для новой группы
4. **MPI_Comm_rank**: Получаем новый номер *rank* процесса в новом коммуникаторе
5. Выполняем передачу данных
6. **MPI_Comm_free и MPI_Group_free**: Освобождаем описатели нового коммуникатора и новой группы

Схема коммуникатор-группа



Работа с процессами в группе

- Определение числа процессов в группе:
  ```
  MPI_Group_size(MPI_Group group,int *size)
  ```
- Определение номера процесса в группе:
  ```
  MPI_Group_rank(MPI_Group group,int *rank)
  ```
- Установка соответствия между номерами процессов в двух группах:
  ```
  MPI_Group_translate_ranks( MPI_Group group1,
  int n, int *ranks1, MPI_Group group2, int
  *ranks2)
  ```

Создание групп

- `MPI_Comm_group(MPI_Comm comm, MPI_Group *group)`
- `MPI_Group_union( MPI_Group group1, MPI_Group group2, MPI_Group *newgroup)`
- `MPI_Group_intersection(MPI_Group group1, MPI_Group group2, MPI_Group *newgroup)`
- `MPI_Group_difference(MPI_Group group1, MPI_Group group2, MPI_Group *newgroup)`

Создание групп

- `MPI_Group_incl(MPI_Group group, int n, int *ranks, MPI_Group *newgroup)`
- `MPI_Group_excl(MPI_Group group, int n, int *ranks, MPI_Group *newgroup)`
- `MPI_Group_range_incl( MPI_Group group, int n, int ranges[][3], MPI_Group *newgroup)`
- `MPI_Group_range_excl( MPI_Group group, int n, int ranges[][3], MPI_Group *newgroup)`

Уничтожение созданных групп

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

Функции доступа к коммуникатору

- ```
  int MPI_Comm_size(
  MPI_Comm comm, int *size)
  ```

- ```
  int MPI_Comm_rank(
  MPI_Comm comm, int *rank)
  ```

Сравнение двух коммуникаторов

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

Возможные значения результата сравнения:

- MPI_IDENT (один и тот же объект)
- MPI_CONGRUENT (две области связи, одни и те же атрибуты группы)
- MPI_SIMILAR (другое упорядочивание групп)
- MPI_UNEQUAL

Создание и дублирование коммуникатора

- ```
  MPI_Comm_dup(MPI_Com
  m comm,   MPI_Comm
  *newcomm)
  ```

- ```
  MPI_Comm_create(MPI_
  Comm comm,
  MPI_Group group,
  MPI_Comm *newcomm)
  ```

Расщепление коммуникатора

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

Уничтожение коммуникатора

```
MPI_Comm_free(MPI_Comm *comm)
```