

Технология программирования MPI (4)

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

Суперкомпьютерная академия МГУ
Москва, 2020

МРІ

Пересылка разнотипных данных

MPI

Сообщение – массив однотипных данных, расположенных в последовательных ячейках памяти.

Для пересылки разнотипных данных можно использовать:

- Производные типы данных
- Упаковку данных

MPI

Производные типы данных

MPI

Производные типы данных создаются во время выполнения программы с помощью подпрограмм-конструкторов.

Создание типа:

- Конструирование типа
- Регистрация типа

MPI

Производный тип данных характеризуется последовательностью базовых типов и набором значений смещения относительно начала буфера обмена.

Смещения могут быть как положительными, так и отрицательными, не требуется их упорядоченность.

MPI

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

Создание нового типа данных **newtype**, состоящего из **count** последовательно расположенных элементов базового типа данных **type**.

```
MPI_Type_contiguous(5, MPI_INT, &newtype);
```

MPI

```
int MPI_Type_vector(int count,  
int blocklen, int stride,  
MPI_Datatype type, MPI_Datatype  
*newtype)
```

Создание нового типа данных **newtype**, состоящего из **count** блоков по **blocklen** элементов базового типа данных **type**.

Следующий блок начинается через **stride** элементов после начала предыдущего.

MPI

```
count=2;  
blocklen=3;  
stride=5;  
MPI_Type_vector(count, blocklen, stride,  
MPI_DOUBLE, &newtype);
```

Создание нового типа данных (тип элемента, количество элементов от начала буфера):

```
{(MPI_DOUBLE, 0), (MPI_DOUBLE, 1),  
(MPI_DOUBLE, 2),  
 (MPI_DOUBLE, 5), (MPI_DOUBLE, 6),  
(MPI_DOUBLE, 7)}
```

MPI

```
int MPI_Type_create_hvector(int  
count, int blocklen, MPI_Aint  
stride, MPI_Datatype type,  
MPI_Datatype *newtype)
```

Создание нового типа данных **newtype**, состоящего из **count** блоков по **blocklen** элементов базового типа данных **type**.

Следующий блок начинается через **stride** байт после начала предыдущего.

MPI

```
int  
MPI_Type_create_indexed_block(  
int count, int blocklen, int  
displs[], MPI_Datatype type,  
MPI_Datatype *newtype)
```

Создание нового типа данных **newtype**, состоящего из **count** блоков по **blocklen** элементов базового типа данных **type**.

Смещения блоков с начала буфера отправки в количестве элементов базового типа данных **type** задаются в массиве **displs**.

MPI

```
int MPI_Type_indexed(int count,  
int *blocklens, int *displs,  
MPI_Datatype type, MPI_Datatype  
*newtype)
```

Создание нового типа данных **newtype**, состоящего из **count** блоков по **blocklens[i]** элементов базового типа данных. **i**-ый блок начинается через **displs[i]** элементов с начала буфера.

MPI

```
for(i=0; i<n; i++){  
    blocklens[i]=n-i;  
    displs[i]=(n+1)*i;  
}  
MPI_Type_indexed(n, blocklens, displs,  
MPI_DOUBLE, &newtype)
```

Создание нового типа данных для описания верхнетреугольной матрицы.

MPI

```
int MPI_Type_create_hindexed(int  
count, int *blocklens, MPI_Aint  
*displs, MPI_Datatype type,  
MPI_Datatype *newtype)
```

Создание нового типа данных **newtype**, состоящего из **count** блоков по **blocklens[i]** элементов базового типа данных. **i**-ый блок начинается через **displs[i]** байт с начала буфера.

MPI

```
int MPI_Type_create_struct(int  
count, int *blocklens, MPI_Aint  
*displs, MPI_Datatype *types,  
MPI_Datatype *newtype)
```

Создание структурного типа данных из **count** блоков по **blocklens[i]** элементов типа **types[i]**. **i**-ый блок начинается через **displs[i]** байт с начала буфера.

MPI

```
blocklens[0]=3;  
blocklens[1]=2;  
types[0]=MPI_DOUBLE;  
types[1]=MPI_CHAR;  
displs[0]=0;  
displs[1]=24;  
MPI_Type_create_struct(2, blocklens, displs,  
types, &newtype);
```

Создание нового типа данных (тип элемента, количество байт от начала буфера):

```
{(MPI_DOUBLE, 0), (MPI_DOUBLE, 8),  
(MPI_DOUBLE, 16),  
(MPI_CHAR, 24), (MPI_CHAR, 25)}
```


MPI

```
int MPI_Type_create_subarray(int  
ndims, int sizes[], int  
subsizes[], int starts[], int  
order, MPI_Datatype type,  
MPI_Datatype *newtype)
```

newtype задаёт **ndims**-мерный подмассив исходного **ndims**-мерного массива. **sizes** задает размеры по каждому измерению исходного массива, **subsizes** – размеры по каждому измерению выделяемого подмассива.

MPI

starts задаёт стартовые координаты каждого измерения выделяемого подмассива в исходном массиве. Все массивы индексируются с 0. Задаваемые значения не должны выводить подмассив за пределы исходного массива ни по одному из измерений. **order** задаёт порядок хранения элементов многомерного массива: **MPI_ORDER_C** (по строкам), **MPI_ORDER_FORTRAN** (по столбцам). **type** задаёт тип элементов массива.

MPI

```
MPI_Datatype newtype;  
int sizes[2], subsizes[2], starts[2];  
int rank;  
MPI_Comm_rank(MPI_COMM_WORLD, &rank);  
sizes[0] = 100;  
sizes[1] = 100;  
subsizes[0] = 100;  
subsizes[1] = 25;  
starts[0] = 0;  
starts[1] = rank*subsizes[1];  
MPI_Type_create_subarray(2, sizes, subsizes,  
starts, MPI_ORDER_C, MPI_DOUBLE, &newtype);
```

MPI

```
int MPI_Type_create_darray(int  
size, int rank, int ndims, int  
gsize[], int distribs[], int  
dargs[], int psize[], int  
order, MPI_Datatype type,  
MPI_Datatype *newtype)
```

Создает новый тип данных **newtype**, представляющий собой **ndims**-мерный массив элементов типа **type**, распределенный по **ndims**-мерной решетке процессов.

MPI

size задает общее число процессов, а массив **psizes** - количество процессов по каждому измерению. Произведение всех элементов **psizes** должно быть равным **size**. Процессы в создаваемой решетке распределяются построчно. В массиве **gsizes** задается число элементов распределяемого массива по каждому из измерений.

MPI

В массиве **distribs** для каждой размерности задается способ распределения элементов массива; возможные значения:

- **MPI_DISTRIBUTE_BLOCK** (блочное распределение);
- **MPI_DISTRIBUTE_CYCLIC** (циклическое распределение);
- **MPI_DISTRIBUTE_NONE** (данное измерение не распределяется).

MPI

В массиве **dargs** задаются параметры распределения по каждому из измерений (количество элементов в одной порции распределяемых данных). Если используется распределение **MPI_DISTRIBUTE_NONE**, то параметр распределения игнорируется. Для блочного распределения блоками данных максимально возможного размера в качестве параметра задается predeterminedенная константа **MPI_DISTRIBUTE_DFLT_DARG**.

MPI

При использовании блочного распределения должны быть распределены все элементы по данному измерению (должно выполняться $dargs(i) * psize(i) \geq gsize(i)$).

Если при циклическом распределении используется константа

MPI_DISTRIBUTE_DFLT_DARG, то

распределение идет по одному элементу.

Параметр **order** задает порядок хранения элементов многомерного массива: либо

MPI_ORDER_C (по строкам), либо

MPI_ORDER_FORTRAN (по столбцам).

MPI

```
int MPI_Type_commit(MPI_Datatype  
*datatype)
```

Регистрация созданного производного типа данных **datatype**. После регистрации этот тип данных можно использовать в операциях обмена. Предопределенные типы данных регистрировать не нужно.

MPI

```
int MPI_Type_free (MPI_Datatype  
*datatype)
```

Аннулирование производного типа данных `datatype`.

`datatype` устанавливается в значение `MPI_DATATYPE_NULL`.

Производные от `datatype` типы данных остаются. Предопределённые типы данных не могут быть аннулированы.

MPI

```
int MPI_Type_dup (MPI_Datatype  
type, MPI_Datatype *newtype)
```

Создает новый тип данных **newtype**, аналогичный типу данных **type**. При этом производится копирование всех ассоциированных с типом **type** атрибутов. Если исходный тип данных уже был зарегистрирован, то автоматически будет зарегистрирован и создаваемый тип данных.

MPI

```
int MPI_Get_address(void  
*location, MPI_Aint *address)
```

Определение абсолютного байт-адреса **address** размещения массива **location** в оперативной памяти компьютера. Адрес отсчитывается от некоторого базового адреса, значение которого содержится в системной константе **MPI_BOTTOM**.

MPI

```
blocklens[0] = 1;
blocklens[1] = 1;
types[0] = MPI_DOUBLE;
types[1] = MPI_CHAR;
MPI_Get_address(dat1, &displs[0]);
MPI_Get_address(dat2, &displs[1]);
MPI_Type_create_struct(2, blocklens, displs,
types, &newtype);
MPI_Type_commit(&newtype);
MPI_Send(MPI_BOTTOM, 1, newtype, dest, tag,
MPI_COMM_WORLD);
```

MPI

```
int MPI_Type_size(MPI_Datatype  
datatype, int *size)
```

Определение размера типа **datatype** в байтах (объёма памяти, занимаемого одним элементом данного типа).

MPI

```
int MPI_Type_get_extent(  
MPI_Datatype datatype, MPI_Aint  
*lb, MPI_Aint *extent)
```

Для элемента типа данных **datatype** определяет смещение от начала буфера данных нижней границы **lb** и диапазон **extent** (разницу между верхней и нижней границами) в байтах.

МРІ

Упаковка данных

MPI

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

Упаковка **incount** элементов типа **datatype** из массива **inbuf** в массив **outbuf** со сдвигом **position** байт. **outbuf** должен содержать хотя бы **outsize** байт.

MPI

Параметр **position** увеличивается на число байт, равное размеру записи.

Параметр **comm** указывает на коммуникатор, в котором в дальнейшем будет пересылаться сообщение.

Для пересылки упакованных данных используется тип данных **MPI_PACKED**.

MPI

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

Распаковка из массива **inbuf** со сдвигом **position** байт в массив **outbuf** **outcount** элементов типа **datatype**.
Массив **inbuf** имеет размер не менее **insize** байт.

MPI

```
int MPI_Pack_size(int incount,  
MPI_Datatype datatype, MPI_Comm  
comm, int *size)
```

Определение необходимого объёма памяти (в байтах) для упаковки **incount** элементов типа **datatype**. Необходимый для упаковки размер может превышать сумму размеров пакуемых элементов данных.

MPI

```
#include <stdio.h>
#include "mpi.h"
int main(int argc, char **argv)
{
    int size, rank, position, i;
    float a[10];
    char b[10], buf[100];
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    for(i = 0; i<10; i++){
        a[i] = rank + 1.0;
        if(rank==0) b[i]='a';
        else b[i] = 'b';
    }
    position=0;
```

MPI

```
if(rank==0) {
    MPI_Pack(a, 10, MPI_FLOAT, buf, 100, &position,
MPI_COMM_WORLD);
    MPI_Pack(b, 10, MPI_CHAR, buf, 100, &position,
MPI_COMM_WORLD);
    MPI_Bcast(buf, 100, MPI_PACKED, 0, MPI_COMM_WORLD);
} else{
    MPI_Bcast(buf, 100, MPI_PACKED, 0, MPI_COMM_WORLD);
    MPI_Unpack(buf, 100, &position, a, 10, MPI_FLOAT,
MPI_COMM_WORLD);
    MPI_Unpack(buf, 100, &position, b, 10, MPI_CHAR,
MPI_COMM_WORLD);
}
for(i = 0; i<10; i++) printf("process %d a=%f b=%c\n",
rank, a[i], b[i]);
MPI_Finalize();}
```

MPI

Производные типы данных:

- сложнее использовать;
- + не нужны дополнительные буфера;
- + нет дополнительных копирований.

Упаковка данных:

- + проще использовать;
- требуется дополнительный буфер;
- требуются дополнительные копирования.

MPI

Задание 7: Напишите программу на MPI, в которой все процессы приложения пересылают нулевому процессу ранг процесса и название узла (полученное с помощью вызова процедуры **MPI_Get_processor_name**), на котором данный процесс запущен. Реализуйте два варианта пересылок: при помощи структурного типа данных и при помощи упаковки/распаковки.