

Многопоточные вычисления на основе технологий MPI и OpenMP: Типы данных

Н. И. Хохлов

МФТИ, Долгопрудный

19 октября 2016 г.

- Типы данных используются при передачи сообщений между процессами.
- Есть готовые константы для встроенных типов данных языка.
- Позволяет создавать собственные типы данных, в том числе для данных не лежащих в памяти последовательно.

Типы данных для языка C

Тип C	Тип MPI
int	MPI_INT
float	MPI_FLOAT
char	MPI_CHAR
double	MPI_DOUBLE
long	MPI_LONG
long double	MPI_LONG_DOUBLE
*	MPI_BYTE
*	MPI_PACKED

- Тип данных в MPI это объект, состоящий из базовых типов (types) и отступов (disp) в байтах для каждого из типов.
- Отспуты рассматриваются относительно отсылаемого буфера.
- Тип данных рассматривается как последовательность пар базовый тип-отступ.
- Данный объект называется typemap.

$$Typemap = \{(type_0, disp_0), (type_1, disp_1), \dots (type_{n-1}, disp_{n-1})\}.$$

Example (MPI_INT)

$$Typemap = (int, 0).$$

Type signature

- Сигнатура типа (type signature) состоит из последовательности базовых типов в данном типе.
- Используется MPI для понимания как интерпретировать конкретные данные по заданном отступу.
- Отступы (disp) говорят MPI где брать данные для пересылки или куда их складывать при приеме.

$$\textit{Type signature} = \{type_0, type_1, \dots type_{n-1}\}.$$

Введем

$$lb(\text{Typemap}) = \min_j(\text{disp}_j),$$

$$ub(\text{Typemap}) = \max_j(\text{disp}_j + \text{sizeof}(\text{type}_j)) + \text{pad},$$

$$\text{extent}(\text{Typemap}) = ub(\text{Typemap}) - lb(\text{Typemap}).$$

lb – нижняя граница (lower bound)

Начало отступов для типа, можно рассматривать как адрес первого байта начала данных для данного типа.

ub – верхняя граница (upper bound)

Конец типа, положение последнего байта для данного типа.

extent – размер типа

Разница верхней и нижней границ типа, может быть выровнена по памяти.

sizeof – размер базового типа в байтах.

Рассмотрим значение параметра **pad**.

Его использование связано с особенностью выравнивания памяти на конкретной машине. Такие языки как C или Fortran требуют, чтобы типы данных, которые занимают несколько байт располагались только в определенных местах в памяти (memory alignment).

Обычно требуется, чтобы адрес начала типа был кратен его размеру.

Рассмотрим тип

$$\text{Typemap}_1 = \{(int, 0), (char, 4)\},$$

тогда

$$lb(\text{Typemap}_1) = \min(0, 4) = 0,$$

$$ub(\text{Typemap}_1) = \max(0 + 4, 4 + 1) = 5.$$

Однако следующий `int` может лежать только через 8 байт, поэтому $pad = 3$ и

$$extent(\text{Typemap}_1) = 8.$$

(На рассматриваемой архитектуре).

- `int MPI_Type_extent(MPI_Datatype datatype, MPI_Aint *extent)`
– получить `extent` для данного типа.
 - `datatype` – тип данных.
 - `extent` – возвращаемое значение (специальный целочисленный тип).
- `int MPI_Type_lb(MPI_Datatype datatype, MPI_Aint *displacement)` – получить нижнюю границу для данного типа.
 - `datatype` – тип данных.
 - `displacement` – возвращаемое значение (специальный целочисленный тип).
- `int MPI_Type_ub(MPI_Datatype datatype, MPI_Aint *displacement)` – получить верхнюю границу для данного типа.
 - `datatype` – тип данных.
 - `displacement` – возвращаемое значение (специальный целочисленный тип).

- `int MPI_Type_size(MPI_Datatype datatype, int *size)` – получить размер для данного типа.
 - `datatype` – тип данных.
 - `size` – возвращаемое значение.

Отличие между `size` и `extent` в том, что `size` возвращает фактический размер типа (сколько он занимает байт), а `extent` то сколько он занимает места в памяти.

Использование `typemap` позволяет создать произвольные типы данных, однако напрямую работать с ним не очень удобно. В MPI есть функционал для работы с часто используемыми типами данных.

- **Contiguous** – создает новый тип путем последовательного повторения старого типа друг за другом.
- **Vector** – создает новый тип путем последовательного повторения старого типа друг за другом с заданным отступом, задающимся в `extent` старого типа.
- **HVector** – тоже самое, что `Vector`, но размер отступа может быть не кратен `extent` старого типа.
- **Indexed** – тоже самое, что `Vector`, но отступ для каждого элемента задается отдельно в `extent` старого типа.
- **HIndexed** – тоже самое, что `HIndexed`, но отступ задается в байтах.
- **Struct** – задание произвольных типов в виде `typemap`.

MPI_Type_contiguous

Создает новый тип путем последовательного повторения старого.
Каждый новый элемент имеет отступ в extent старого типа.

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

- **count** – число старых типов в новом типе;
- **oldtype** – старый (базовый) тип данных;
- **newtype** – новый тип данных.

Пример 1

```
int a[4][4] = {1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16};  
...  
MPI_Type_contiguous(4, MPI_INT, &rowt);  
...  
MPI_Send(&a[2][0], rowt, 1, ... );
```

a[4][4]

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

1 элемент rowt &a[2][0]

9	10	11	12
---	----	----	----

Пример 2

Старый тип

$$\{(double, 0), (char, 8)\},$$

extent = 16, count = 3, тогда новый тип

$$\{(double, 0), (char, 8), (double, 16),$$

$$(char, 24), (double, 32), (char, 40)\}.$$

Старый тип

$$\{(type(0), disp(0)), \dots, (type(n-1), disp(n-1))\}$$

extent = ex, count = n, тогда новый тип

$$\begin{aligned} &\{(type(0), disp(0)), \dots, (type(n-1), disp(n-1)), \\ &\quad (type(0), disp(0) + ex), \dots, (type(n-1), \\ &\quad disp(n-1) + ex), \dots, (type(0), disp(0) + ex * (count - 1)), \\ &\quad \dots, (type(n-1), disp(n-1) + ex * (count - 1))\}. \end{aligned}$$

Последовательно задает расположение старого типа в памяти через одинаковые отступы одинаковыми блоками. Размер отступа задается в старых типах.

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

- **count** – количество блоков;
- **blocklength** – число старых типов в одном блоке;
- **stride** – расстояние в старых типах между началами соседних блоков;
- **oldtype** – старый (базовый) тип данных;
- **newtype** – новый тип данных.

Пример 1

```
int a[4][4] = {1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16};  
...  
MPI_Type_vector(4,1,4,MPI_INT,&colt);  
...  
MPI_Send(&a[0][1], colt, 1, ... );
```

a[4][4]

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

1 элемент colt &a[0][1]

2	6	10	14
---	---	----	----

Пример 2

Старый тип

$$\{(double, 0), (char, 8)\},$$

extent = 16, count = 2, blocklength = 3, stride = 4 тогда новый тип

$$\{(double, 0), (char, 8), (double, 16), (char, 24), (double, 32), (char, 40), \\ (double, 64), (char, 72), (double, 80), (char, 88), (double, 96), (char, 104)\}.$$

Пример 3

Старый тип

$$\{(double, 0), (char, 8)\},$$

extent = 16, count = 3, blocklength = 1, stride = -2 тогда новый тип

$$\{(double, 0), (char, 8),$$
$$(double, -32), (char, -24),$$
$$(double, -64), (char, -56)\}.$$

В общем случае

Старый тип $\{(type(0), disp(0)), \dots, (type(n-1), disp(n-1))\}$ extent = ex, count = n, blocklength = bl, stride = stride, тогда новый тип

$$\begin{aligned} & \{(type(0), disp(0)), \dots, (type(n-1), disp(n-1)), \\ & (type(0), disp(0) + ex), \dots, (type(n-1), disp(n-1) + ex), \dots, \\ & (type(0), disp(0) + (bl - 1) * ex), \dots, \\ & (type(n-1), disp(n-1) + (bl - 1) * ex), \\ & (type(0), disp(0) + stride * ex), \dots, (type(n-1), \\ & disp(n-1) + stride * ex), \dots, \\ & (type(0), disp(0) + (stride + bl - 1) * ex), \dots, \\ & (type(n-1), disp(n-1) + (stride + bl - 1) * ex), \dots, \\ & (type(0), disp(0) + stride * (count - 1) * ex), \dots, \\ & (type(n-1), disp(n-1) + stride * (count - 1) * ex), \dots, \\ & (type(0), disp(0) + (stride * (count - 1) + bl - 1) * ex), \dots, \\ & (type(n-1), disp(n-1) + (stride * (count - 1) + bl - 1) * ex)\} \end{aligned}$$

Последовательно задает расположение старого типа в памяти через одинаковые отступы одинаковыми блоками. Размер отступа задается в байтах.

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

- **count** – количество блоков;
- **blocklength** – число старых типов в одном блоке;
- **stride** – расстояние в байтах между началами соседних блоков;
- **oldtype** – старый (базовый) тип данных;
- **newtype** – новый тип данных.

MPI_Type_indexed

Последовательно задает расположение старого типа в памяти через заданные отступы блоками заданных размеров. Размер отступа задается в старых типах.

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

- **count** – количество блоков;
- **array_of_blocklengths** – массив размеров блоков (в старых типах);
- **array_of_displacements** – массив отступов от начала данных (в старых типах);
- **oldtype** – старый (базовый) тип данных;
- **newtype** – новый тип данных.

Пример 1

```
int a[16] = {1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16};  
...  
int count = 2;  
int bl[2] = {4,2};  
int disp[2] = {5,12};  
MPI_Type_indexed(count,bl,disp,MPI_INT,&newt)  
...  
MPI_Send(a, newt, 1, ... );
```

a[16]

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----

1 элемент newt a

6	7	8	9	13	14
---	---	---	---	----	----

Пример 2

Старый тип

$$\{(double, 0), (char, 8)\},$$

extent = 16, count = 2, B = (3, 1), D = (4, 0) тогда новый тип

$$\{(double, 64), (char, 72), (double, 80), (char, 88), (double, 96), (char, 104), \\ (double, 0), (char, 8)\}.$$

В общем случае

Старый тип $\{(type(0), disp(0)), \dots, (type(n-1), disp(n-1))\}$ extent = ex, count = n, размеры блоков B, отступы D, тогда новый тип

$$n \times S^{count} - 1$$

$$i = 0, B[i] :$$

$$\{(type(0), disp(0) + D[0] * ex), \dots,$$

$$(type(n-1), disp(n-1) + D[0] * ex), \dots,$$

$$(type(0), disp(0) + (D[0] + B[0] - 1) * ex), \dots,$$

$$(type(n-1), disp(n-1) + (D[0] + B[0] - 1) * ex), \dots,$$

$$(type(0), disp(0) + D[count - 1] * ex), \dots,$$

$$(type(n-1), disp(n-1) + D[count - 1] * ex), \dots,$$

$$(type(0), disp(0) + (D[count - 1] + B[count - 1] - 1) * ex), \dots,$$

$$(type(n-1), disp(n-1) + (D[count - 1] + B[count - 1] - 1) * ex)\}.$$

Вызов

```
MPI_Type_vector(count, blocklength, stride, oldtype, newtype)
```

аналогичен

```
MPI_Type_indexed(count, B, D, oldtype, newtype)
```

с параметрами

$$D[j] = j * \text{stride}, j = 0, \dots, \text{count}-1$$
$$B[j] = \text{blocklength}, j = 0, \dots, \text{count}-1$$

MPI_Type_hindexed

Последовательно задает расположение старого типа в памяти через заданные отступы блоками заданных размеров. Размер отступа задается в байтах.

```
int MPI_Type_hindexed(int count, int array_of_blocklengths[],  
MPI_Aint array_of_displacements[], MPI_Datatype oldtype,  
MPI_Datatype *newtype);
```

- **count** – количество блоков;
- **array_of_blocklengths** – массив размеров блоков (в старых типах);
- **array_of_displacements** – массив отступов от начала данных (в байтах);
- **oldtype** – старый (базовый) тип данных;
- **newtype** – новый тип данных.

MPI_Type_struct

Последовательно задает расположение набора старых типов в памяти через заданные отступы блоками заданных размеров. Размер отступа задается в байтах.

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

- **count** – количество блоков;
- **array_of_blocklengths** – массив размеров блоков (в старых типах);
- **array_of_displacements** – массив отступов от начала данных (в байтах);
- **array_of_types** – массив старых типов;
- **newtype** – новый тип данных.

Пример 1

```
struct {  
    int type;  
    double x, y, z;  
} point;  
...  
point p;  
...  
int count = 2;  
int B[2] = {1,3};  
MPI_Datatype T[2] = {MPI_INT, MPI_DOUBLE};  
MPI_Aint D[2] = 0, extent(MPI_INT);  
MPI_Type_struct(3, B, D, T, newt);  
...  
MPI_Send(&p, newt, 1, ... );
```

Пример 1

Новый тип

MPI_INT

MPI_DOUBLE

MPI_DOUBLE

MPI_DOUBLE

Пример 2

Старый тип

$$type1 = \{(double, 0), (char, 8)\},$$

extent1 = 16, B = (2, 1, 3), D = (0, 16, 26), T = (MPI_FLOAT, type1, MPI_CHAR), тогда

$$\{(float, 0), (float, 4), (double, 16), (char, 24), \\ (char, 26), (char, 27), (char, 28)\}.$$

Проблема пересылки нескольких структур и extent

```
struct {  
    int a;  
    char b;} foo;
```

`sizeof(foo) = sizeof(int) + sizeof(char);`

Правильно делать так (при пересылке нескольких структур):

```
blen[0] = 1; indices[0] = 0; oldtypes[0] = MPI_INT;  
blen[1] = 1; indices[1] = &foo.b - &foo; oldtypes[1] = MPI_CHAR;  
blen[2] = 1; indices[2] = sizeof(foo); oldtypes[2] = MPI_UB;  
MPI_Type_struct( 3, blen, indices, oldtypes, &newtype );
```

Поскольку $\text{extent} = 2 * \text{sizeof(int)}$.

Должный совпадать сигнатуры типов.

Последовательность работы с типами

- Создать новый тип данных.
- Вызвать `MPI_Type_commit`, после этого можно использовать тип в пересылках.
- Работать с типом в пересылках.
- Вызвать `MPI_Type_free`, для высвобождения памяти.

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

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

```
MPI_Datatype t;  
...  
создаем тип  
...  
MPI_Type_commit(&t);  
...  
пересылки  
...  
MPI_Type_free(&t);
```

Спасибо за внимание! Вопросы?