Параллельное программирование

Вощинская Гильда Эдгаровна

Типы, определяемые пользователем

Все MPI-функции, которые имеют аргумент datatype

(тип данных)использовать в качестве типа определяемый пользователем (или «производный») тип. Они - не «типы» в смысле языка программирования. Они – только «типы», которые MPI делает с помощью функций строителей типов, описывающих расположение в памяти наборов примитивных типов.

Производный тип данных -

непрозрачный объект, который определяет два предмета:

- **последовательность примитивных типов и,**
- **последовательность целого числа смещений (в количестве байт) значений этих типов от начального адреса.**

Не требуется, чтобы смещения были положительными, различными, или в увеличивающемся порядке. Такая пара последовательностей (или последовательность пар) называется *отображением типа.*

Последовательность примитивных типов данных

(смещения игнорируются) называется *сигнатурой типа данных*.

Typemap={(type0, disp0), . . ., (typen,dispn)} -отображение типа,где typei -примитивные типы и dispi - смещения значений этих типов относительно базового адреса.

Typesig={type0, …. typen} соответствующая сигнатура типа.

Это отображение типа, вместе с базовым адресом buf, определяет коммуникационный буфер: коммуникационный буфер, который состоит из n элементов, где i-й элемент стоит по адресу buf+dispi и имеет тип typei.

Строители типов

Использование производного типа

- в функциях обмена сообщениями можно рассматривать как трафарет, наложенный на область памяти, которая содержит передаваемое или принятое сообщение.

Стандартный сценарий определения и использования производных типов включает следующие шаги:

1. Производный тип строится из предопределенных типов MPI и ранее определенных производных типов с помощью специальных функций-конструкторов MPI_Type_contiguous, MPI_Type_vector, MPI_Type_hvector, MPI_Type_indexed, MPI_Type_hindexed, MPI_Type_struct.

2. Новый производный тип регистрируется вызовом функции MPI_Type_commit. Только после регистрации новый производный тип можно использовать в коммуникационных подпрограммах и при конструировании других типов. Предопределенные типы MPI считаются зарегистрированными.

3. Когда производный тип становится ненужным, он уничтожается функцией MPI_Type_free.

Любой тип данных в MPI имеет две характеристики: протяженность и размер, выраженные в байтах:

1. Протяженность типа определяет, сколько байт переменная данного типа занимает в памяти. Эта величина может быть вычислена как:

   адрес_последней_ячейки_данных – адрес_первой_ячейки_данных + длина_последней_ячейки_данных

   (опрашивается подпрограммой MPI_Type_extent).

2. Размер типа определяет количество реально передаваемых байт в коммуникационных операциях. Эта величина равна сумме длин всех базовых элементов определяемого типа (опрашивается подпрограммой MPI_Type_size).

Для простых типов протяженность и размер совпадают.

Функция MPI_Type_extent определяет протяженность элемента некоторого типа

```
int MPI_Type_extent  (MPI_Datatype datatype,
                      MPI_Aint *extent)
```

Входные параметры:

datatype - тип данных

Выходные параметры:

extent -протяженность элемента заданного типа

Имена типов данных в MPI –

непрозрачные поэтому нужно использовать функцию MPI_Type_extent, чтобы определить размер («size») типа.
Нельзя использовать по аналогии, как в С функцию sizeof (datatype), например, sizeof (MPI_DOUBLE).
Она возвратит размер непрозрачного заголовка, который является размером указателя, и, конечно же, отличается от значения sizeof (double).

Функция MPI_Type_size

определяет "чистый" размер элемента некоторого типа за вычетом пустых промежутков.

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

Входные параметры:

datatype - тип данных.

Выходные параметры:

size - размер элемента заданного типа.

Элементы, которые многократно встречаются в datatype, учитываются с их кратностью.

Для примитивного datatype, эта функция возвращает ту же самую информацию как MPI_Type_extent.

Пример

Допустим datatype имеет тип отображения Type={(double,0),(char,1)}. Тогда запрос к MPI_Type_extent (datatype, i) возвратит i = 16(т.к. double требует выравнивания на границу, кратную 8, 8+1+7); запрос к MPI_Type_size (datatype, i) возвратит i = 9 (8 байт (double) + 1 байт (char) = 9).

Строитель смежных типов данных CONTIGUOUS

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

Входные параметры:

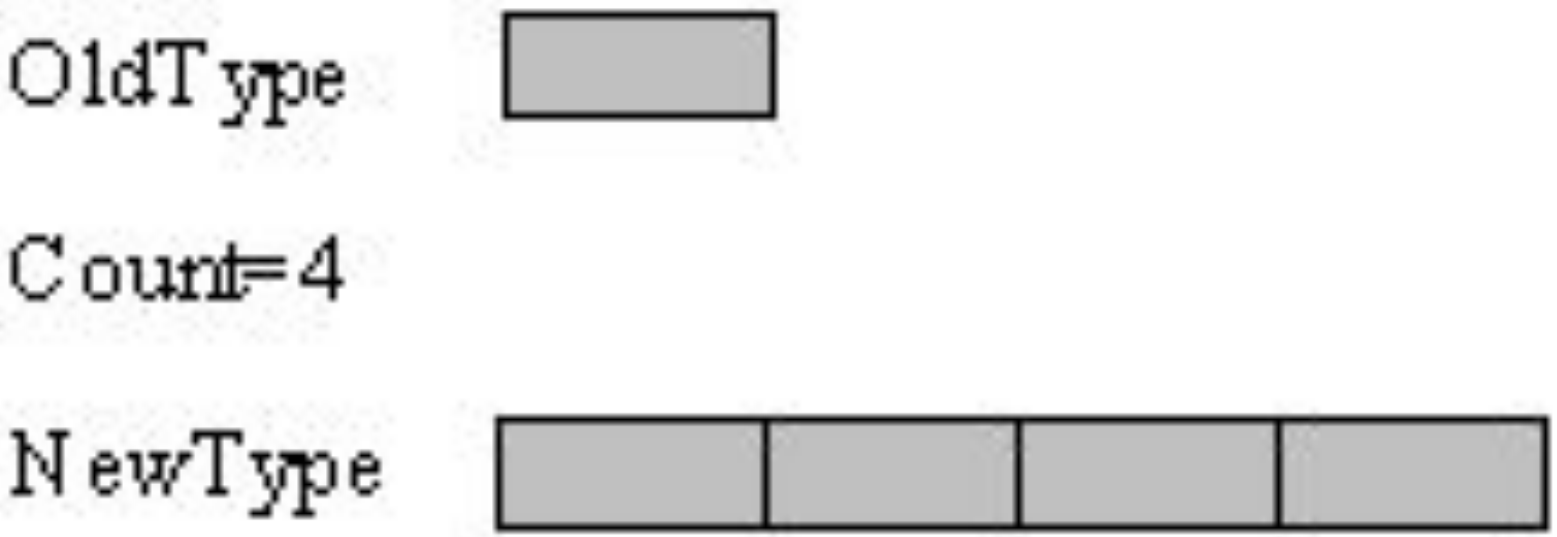
count - число элементов базового типа;

oldtype - базовый тип данных.

Выходные параметры:

newtype -новый производный тип данных.

Графическая интерпретация работы конструктора MPI_Type_contiguous



Векторный строитель типов данных MPI_Type_vector

создает тип, элемент которого представляет собой несколько равноудаленных друг от друга блоков из одинакового числа смежных элементов базового типа.

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

Входные параметры:

count - число блоков;

blocklength - число элементов базового типа в каждом блоке;

stride-шаг между началами соседних блоков, измеренный числом элементов базового типа;

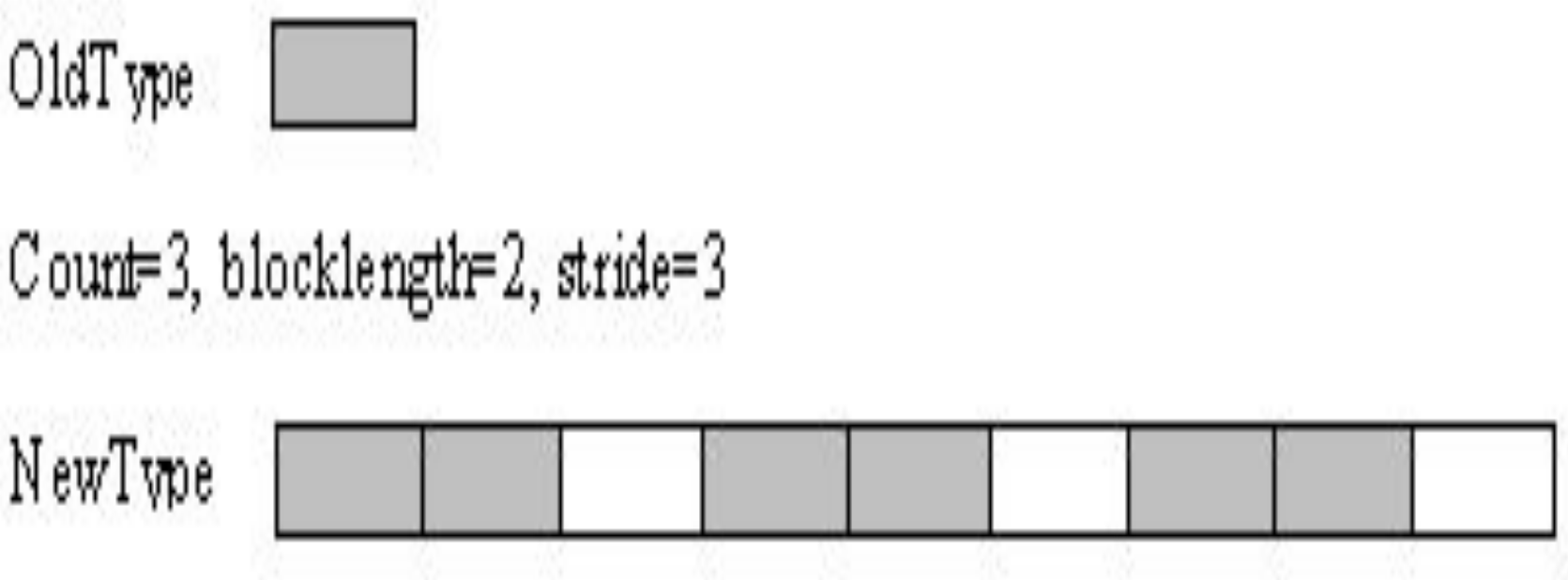
oldtype - базовый тип данных.

Выходные параметры:

newtype - новый производный тип данных.

Функция MPI_Type_vector создает тип

newtype, элемент которого состоит из count блоков, каждый из которых содержит одинаковое число blocklength элементов типа oldtype. Шаг stride между началом блока и началом следующего блока всюду одинаков и кратен протяженности представления базового типа.



Векторный строитель типов данных MPI_Type_hvector

расширяет возможности конструктора MPI_Type_vector, позволяя задавать произвольный шаг между началами блоков в байтах.

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

Входные параметры:

count - число блоков;

blocklength - число элементов базового типа в каждом блоке;

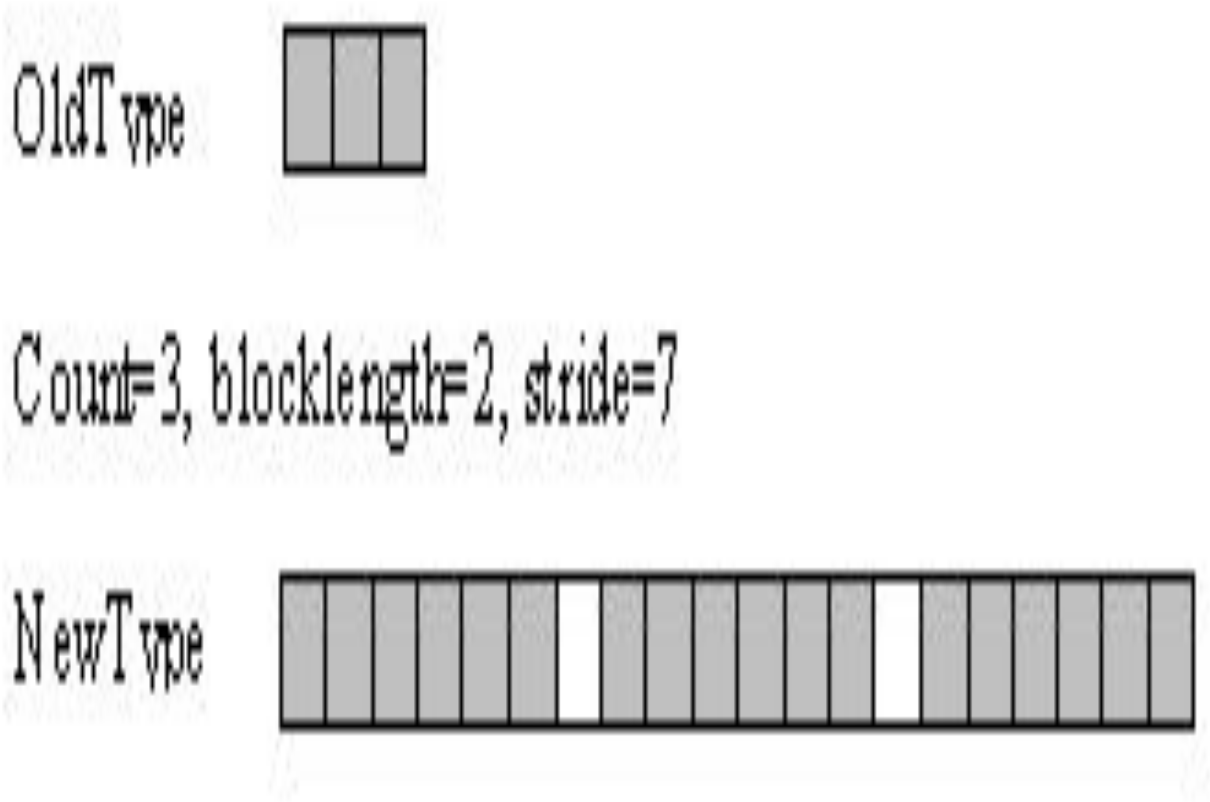
stride - шаг между началами соседних блоков в байтах;

oldtype - базовый тип данных.

Выходные параметры:

newtype - новый производный тип данных.

Графическая интерпретация работы конструктора MPI_Type_hvector



Конструктор типа MPI_Type_indexed

является более универсальным конструктором по сравнению с MPI_Type_vector, так как элементы создаваемого типа состоят из произвольных по длине блоков с произвольным смещением блоков от начала размещения элемента. Смещения измеряются в элементах старого типа.

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

Входные параметры:

count - число блоков;

array_of_blocklengths - массив, содержащий число элементов базового типа в каждом блоке;

array_of_displacements - массив смещений каждого блока от начала размещения элемента нового типа, смещения измеряются числом элементов базового типа;

oldtype - базовый тип данных.

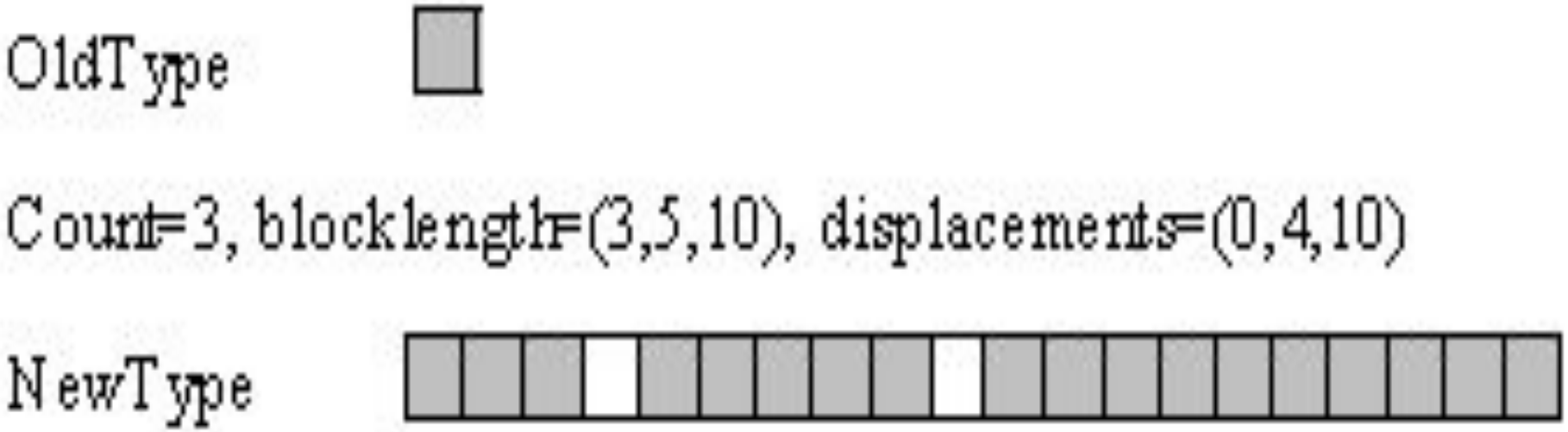
Выходные параметры:

newtype -новый производный тип данных

функция **MPI_Type_indexed**

создает тип newtype, каждый элемент которого состоит из count блоков, где i-ый блок содержит array_of_blocklengths[i] элементов базового типа и смещен от начала размещения элемента нового типа на array_of_displacements[i] элементов базового типа.

Графическая интерпретация работы конструктора MPI_Type_indexed



Конструктор типа MPI_Type_hindexed

идентичен конструктору MPI_Type_indexed
за исключением того, что смещения измеряются в байтах.

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

Входные параметры:

count - число блоков;

array_of_blocklengths - массив, содержащий число элементов базового типа в каждом блоке;

array_of_displacements - массив смещений каждого блока от начала размещения элемента нового типа, смещения измеряются в байтах;

oldtype - базовый тип данных.

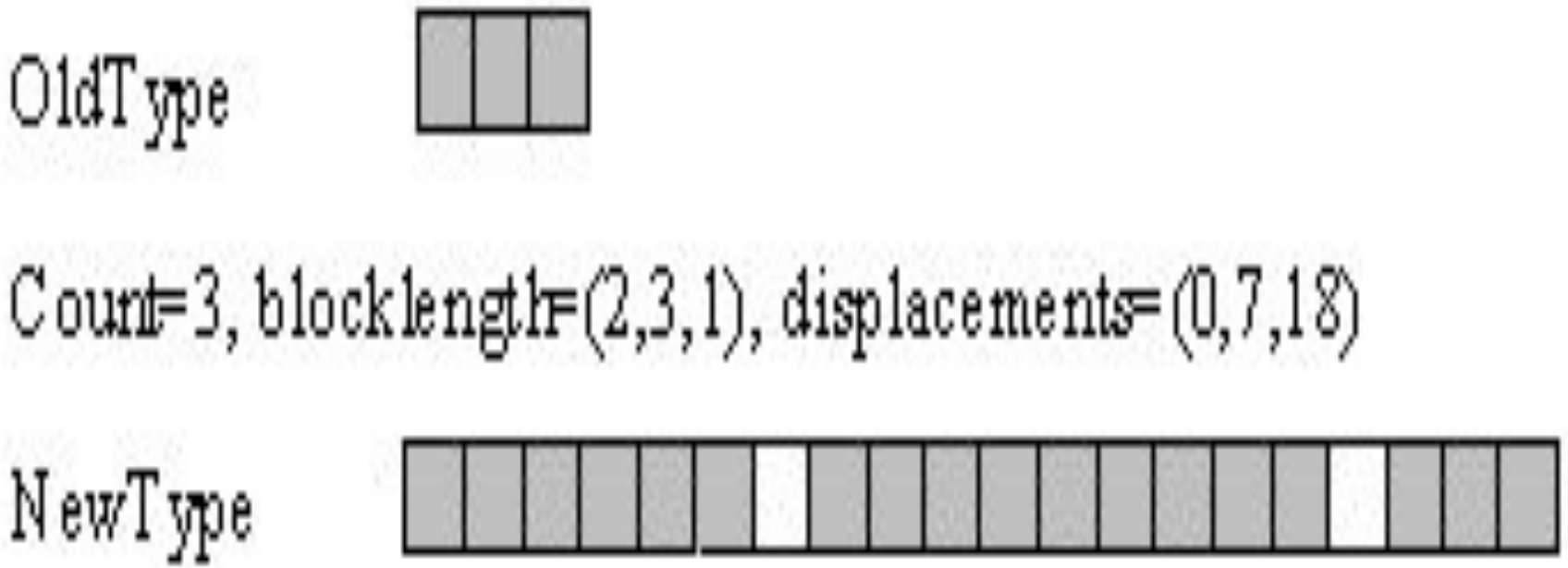
Выходные параметры:

newtype - новый производный тип данных.

Элемент нового типа

состоит из count блоков, где i-ый блок содержит array_of_blocklengths[i] элементов старого типа и смещен от начала размещения элемента нового типа на array_of_displacements[i] байт.

Графическая интерпретация работы конструктора MPI_Type_hindexed:



Конструктор типа MPI_Type_struct

самый универсальный из всех конструкторов типа.

Результат - структура, состоящая из произвольного числа блоков, каждый из которых может содержать произвольное число элементов одного из базовых типов и может быть смещен на произвольное число байтов от начала размещения структуры.

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

Входные параметры:

count - число блоков;

array_of_blocklength - массив, содержащий число элементов одного из базовых типов в каждом блоке;

array_of_displacements - массив смещений каждого блока от начала структуры, смещения измеряются в байтах;

array_of_type - массив, содержащий тип элементов в каждом блоке.

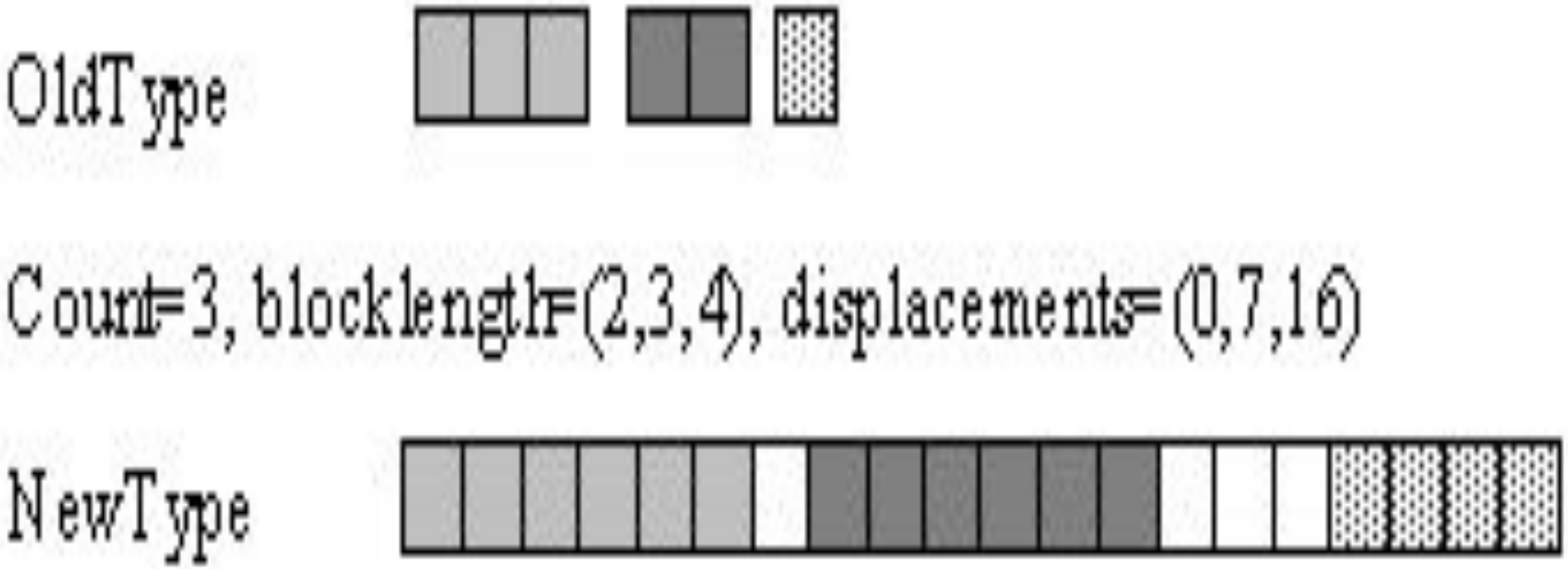
Выходные параметры:

newtype - новый производный тип данных.

Функция MPI_Type_struct

создает тип newtype, элемент которого состоит из count блоков, где i-ый блок содержит array_of_blocklengths[i] элементов типа array_of_types[i]. Смещение i-ого блока от начала размещения элемента нового типа измеряется в байтах и задается в array_of_displacements[i].

Графическая интерпретация работы конструктора MPI_Type_struct:



Использование производных типов данных

Функция MPI_Type_commit регистрирует созданный производный тип. Только после регистрации новый тип может использоваться в коммуникационных операциях.

int MPI_Type_commit(MPI_Datatype *datatype)

Входные параметры:

datatype - новый производный тип данных

Выходные параметры:

datatype - новый производный тип данных.

datatype тип данных, который передается в ОС. Передача не подразумевает, что datatype привязан к текущему содержанию буфера связи. После того, как datatype был передан, он может неоднократно повторно использоваться, чтобы идентифицировать данные.

Освобождение

int MPI_Type_free(MPI_Datatype *datatype)

Входные параметры:

datatype-уничтожаемый производный тип данных.

Выходные параметры:

datatype-уничтожаемый производный тип данных.

MPI_Type_free регистрирует объект типа данных, связанный с datatype для освобождения и устанавливает datatype MPI_DATATYPE_NULL.

Любая связь, которая в это время (постоянно) использует этот datatype, завершится обычно.

Производные типы данных, которые были определены из освобожденного типа данных (datatype), не повреждаются.

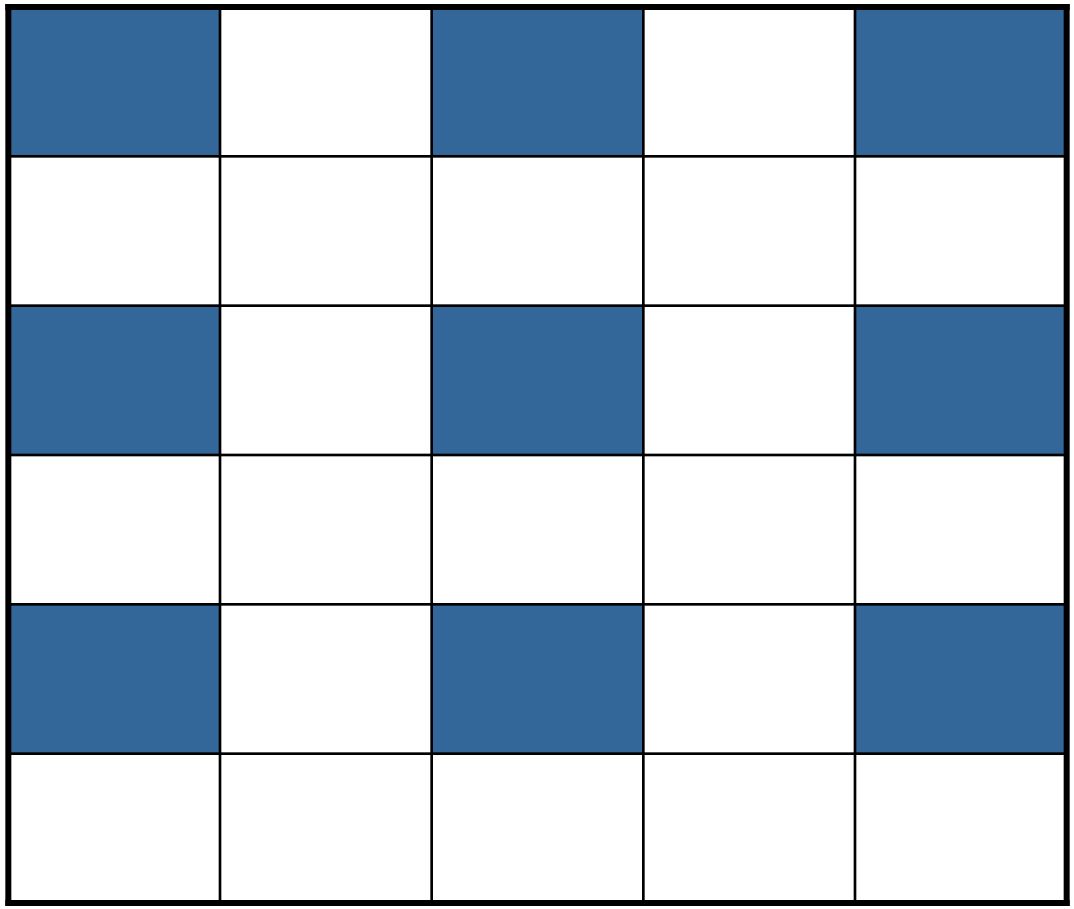
Примеры

Посылка и получение секции 2D (двумерного) массива

Первый запрос к MPI_Type_vector определяет тип данных, который описывает одну строку секции: 1D секция массива, которая состоит из трех чисел float, с расстоянием в два элемента друг от друга.

Второй запрос к MPI_Type_hvector определяет тип данных, который описывает 2D секцию массива: три копии предыдущих 1D секции массива, с расстоянием 10*sizeof(float)

Представление массива в примере2



```
float a[6] [5], e[3] [3];
int oneslice, twoslice, sizeoffloat, myrank;
MPI_status status;
/*  Заштрихованные элементы массива а конструируются в новый
тип  данных  и  помещаются  в  массив  е   */
MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
MPI_Type_extent(MPI_FLOAT, &sizeoffloat);
 /* Создание типа данных для 1D секции   */
MPI_Type_vector(3, 1, 2, MPI_FLOAT, &oneslice);
 /* Создание типа данных для 2D секции */
 MPI_Type_hvector(3, 1,  10*sizeoffloat, oneslice, twoslice);
MPI_Type_commit (twoslice) ;
/*   Передача данных и  прием  на некотором  процессе */
MPI_Sendrecv(a, I, twoslice, myrank, 0, e, 9, MPI_FLOAT,
          myrank, 0, MPI_COMM_WORLD, &status);
```

Передача верхней треугольной матрицы

```
double a[100][100], disp[100], blocklen[100], i;
/* вычисление начала и размера каждой строки
   матрицы, начиная от диагонали*/
for(i=0; i<100; i++)
{
  disp[i]=100*i+i;
  blocklen[i]=100-i;
}
/*создание типа для верхней триангуляции матрицы*/
MPI_Type_indexed(100,blocklen,disp,MPI_DOUBLE,
   &upper);
MPI_Type_commit(&upper);
MPI_Send(a,1,upper,dest,tag,MPI_COMM_WORLD);
```

Транспонирование матрицы

Чтобы транспонировать матрицу, нужно создать первый тип данных из элементов (чисел) строки, отстоящими друг от друга на размер одного измерения матрицы. Затем, создать второй тип из элементов первого типа, и с расстоянием между этими элементами в одно число. После чего матрица посылается со вторым типом данных и принимается матрица теперь уже колонками.

Транспонирование матрицы

```
float a[100][100], b[100][100];
int row, xpose, sizeoffloat, myrank;
MPI_status status;
/*  транспонирование матрицы a в b  */
MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
MPI_Type_extent(MPI_FLOAT, &sizeoffloat);
 /*  создание типа для одной строки - это вектор со 100 float
     элементами и расстоянием равным 100  */
MPI_Type_vector (100, 1, 100, MPI_FLOAT, &row);
 /* создание типа для матрицы упорядоченной по строкам ,
 каждая копия новой строки сдвинута друг относительно друга на
     один элемент)  */
MPI_Type_hvector(100, 1, sizeoffloat, row, &xpose);
 MPI_Type_commit(xpose);
/*  передача матрицы строками и прем ее колонками  */
MPI_Sendrecv(a, 1, xpose, myrank, 0, b, 100*100, MPI_FLOAT,
     myrank, 0, MPI_COMM_WORLD, &status);
```

Освобождение типов

```
int type1, type2;
MPI_Type_contiguous(5, MPI_FLOAT, &type1)
/* создание объекта нового типа */
MPI_Type_commit(type1)
/*  новый  type1 может быть использован для обменов  данными  */
type2=type1
/* type2 может быть использован для обменов данными */
MPI_Type_vector(3, 5, 4, MPT_FLOAT, &type1)
 /* создается объект нового типа */
MPI_Type_commit(type1);
 /*  новый  type1  может  использоваться  для  обменов*/
MPI_Type_free(type2)
/*  освобождение  типа  */
 type2=type1
/*  type2  может  использоваться  для  обменов  */
MPI_Type_free(type2)
 /* type1 и type2 не действительны; type2 имеет величину
    MPI_DATATYPE_NULL  и  type1 не определен */
```