- Какие пересылки данных осуществляются процедурой **`MPI_CART_SHIFT`**?
- Как определить, с какими процессами в топологии графа связан данный процесс?
- Как создать топологию графа, в которой каждый процесс связан с каждым?
- Реализовать разбиение процессов на две группы, в одной из которых осуществляется обмен по кольцу при помощи сдвига в одномерной декартовой топологии, а в другой – коммуникации по схеме master-slave, реализованной при помощи топологии графа.
- Использовать двумерную декартову топологию процессов при реализации параллельной программы перемножения матриц.

Пересылка разнотипных данных

Под *сообщением* в MPI понимается массив однотипных данных, расположенных в последовательных ячейках памяти. Часто в программах требуются пересылки более сложных объектов данных, состоящих из разнотипных элементов или расположенных не в последовательных ячейках памяти. В этом случае можно либо посылать данные небольшими порциями расположенных подряд элементов одного типа, либо использовать копирование данных перед отсылкой в некоторый промежуточный буфер. Оба варианта являются достаточно неудобными и требуют дополнительных затрат как времени, так и оперативной памяти.

Для пересылки разнотипных данных в MPI предусмотрены два специальных способа:

- *Производные типы данных*;
- *Упаковка данных*.

Производные типы данных

Производные типы данных создаются во время выполнения программы с помощью процедур-конструкторов на основе существующих к моменту вызова конструктора типов данных.

Создание типа данных состоит из двух этапов:

1. Конструирование типа.
2. Регистрация типа.

После регистрации производный тип данных можно использовать наряду с предопределенными типами в операциях пересылки, в том числе и в коллективных операциях. После завершения работы с производным типом данных его рекомендуется аннулировать. При этом все произведенные на его основе новые типы данных остаются и могут использоваться дальше.

Производный тип данных характеризуется последовательностью базовых типов данных и набором целочисленных значений смещения элементов типа относительно начала буфера обмена. Смещения могут быть как положительными, так и отрицательными, не обязаны различаться, не требуется их упорядоченность. Таким образом, последовательность элементов данных в производном типе может отличаться от последовательности исходного типа, а один элемент данных может встречаться в конструируемом типе многократно.

```
MPI_TYPE_CONTIGUOUS(COUNT, TYPE, NEWTYPE, IERR)
INTEGER COUNT, TYPE, NEWTYPE, IERR
```

Создание нового типа данных **NEWTYPE**, состоящего из **COUNT** последовательно расположенных элементов базового типа данных **TYPE**. Фактически создаваемый тип данных представляет массив данных базового типа как отдельный объект.

В следующем примере создается новый тип данных **newtype**, который в дальнейшем (после регистрации типа) может использоваться для пересылки пяти расположенных подряд целых чисел.

```
      call MPI_TYPE_CONTIGUOUS(5, MPI_INTEGER, newtype, ierr)
```

```
MPI_TYPE_VECTOR(COUNT, BLOCKLEN, STRIDE, TYPE, NEWTYPE, IERR)
INTEGER COUNT, BLOCKLEN, STRIDE, TYPE, NEWTYPE, IERR
```

Создание нового типа данных **NEWTYPE**, состоящего из **COUNT** блоков по **BLOCKLEN** элементов базового типа данных **TYPE**. Следующий блок начинается через **STRIDE** элементов базового типа данных после начала предыдущего блока.

В следующем примере создается новый тип данных **newtype**, который после регистрации может быть использован для пересылки как единого целого шести элементов данных, которые можно представить следующим образом (тип элемента данных, количество элементов данных от начала буфера посылки):

{(**MPI_REAL**, 0), (**MPI_REAL**, 1), (**MPI_REAL**, 2),
(**MPI_REAL**, 5), (**MPI_REAL**, 6), (**MPI_REAL**, 7)}

```
      count = 2
      blocklen = 3
      stride = 5
      call MPI_TYPE_VECTOR(count, blocklen, stride,
     &                     MPI_REAL, newtype, ierr)
```

```
MPI_TYPE_HVECTOR(COUNT, BLOCKLEN, STRIDE, TYPE, NEWTYPE, IERR)
INTEGER COUNT, BLOCKLEN, STRIDE, TYPE, NEWTYPE, IERR
```

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 блока.

```
MPI_TYPE_INDEXED(COUNT, BLOCKLENS, DISPLS, TYPE, NEWTYPE, IERR)
INTEGER COUNT, BLOCKLENS(*), DISPLS(*), TYPE, NEWTYPE, IERR
```

Создание нового типа данных **NEWTYPE**, состоящего из **COUNT** блоков по **BLOCKLENS(I)** элементов базового типа данных. **I**-й блок начинается через **DISPLS(I)** элементов базового типа данных с начала буфера посылки. Полученный тип данных можно считать обобщением векторного типа.

В следующем примере задается тип данных **newtype** для описания нижнетреугольной матрицы типа **double precision** (при этом учитывается, что в языке Фортран массивы хранятся по столбцам).

```
      do i = 1, n
         blocklens(i) = n-i+1
         displs(i) = n*(i-1)+i-1
      end do
      call MPI_TYPE_INDEXED(n, blocklens, displs,
                            MPI_DOUBLE_PRECISION, newtype, ierr)
```

```
MPI_TYPE_HINDEXED(COUNT, BLOCKLENS, DISPLS, TYPE, NEWTYPE, IERR)
INTEGER COUNT, BLOCKLENS(*), DISPLS(*), TYPE, NEWTYPE, IERR
```

Создание нового типа данных **NEWTYPE**, состоящего из **COUNT** блоков по **BLOCKLENS(I)** элементов базового типа данных. **I**-й блок начинается через **DISPLS(I)** байт с начала буфера посылки.

```
MPI_TYPE_STRUCT(COUNT, BLOCKLENS, DISPLS, TYPES, NEWTYPE, IERR)
INTEGER COUNT, BLOCKLENS(*), DISPLS(*), TYPES(*), NEWTYPE, IERR
```

Создание структурного типа данных **NEWTYPE** из **COUNT** блоков по **BLOCKLENS(I)** элементов типа **TYPES(I)**. **I**-й блок начинается через **DISPLS(I)** байт с начала буфера посылки.

В следующем примере создается новый тип данных **newtype**, который после регистрации может быть использован для пересылки как единого целого пяти элементов данных, которые можно представить следующим образом (тип элемента данных, количество байт от начала буфера посылки):
{(**MPI_DOUBLE_PRECISION**, **0**), (**MPI_DOUBLE_PRECISION**, **8**),

(**MPI_DOUBLE_PRECISION**, **16**), (**MPI_CHARACTER**, **24**), (**MPI_CHARACTER**, **25**)}

```
      blocklens(1) = 3
      blocklens(2) = 2
      types(1) = MPI_DOUBLE_PRECISION
      types(2) = MPI_CHARACTER
      displs(1) = 0
      displs(2) = 24
      call MPI_TYPE_STRUCT(2, blocklens, displs, types,
     &                     newtype, ierr)
```

```
MPI_TYPE_COMMIT(DATATYPE, IERR)
INTEGER DATATYPE, IERR
```

Регистрация созданного производного типа данных **DATATYPE**. После регистрации этот тип данных можно использовать в операциях обмена наравне с предопределенными типами данных. Предопределенные типы данных регистрировать не нужно.

```
MPI_TYPE_FREE(DATATYPE, IERR)
INTEGER DATATYPE, IERR
```

Аннулирование производного типа данных **DATATYPE**. Параметр **DATATYPE** устанавливается в значение **MPI_DATATYPE_NULL**. Гарантируется, что любой начатый обмен, использующий данные аннулируемого типа, будет нормально завершен. При этом производные от **DATATYPE** типы данных остаются и могут использоваться дальше. Предопределенные типы данных не могут быть аннулированы.

```
MPI_TYPE_SIZE(DATATYPE, SIZE, IERR)
INTEGER DATATYPE, SIZE, IERR
```

Определение размера **SIZE** типа данных **DATATYPE** в байтах (объема памяти, занимаемого одним элементом данного типа).

```
MPI_ADDRESS(LOCATION, ADDRESS, IERR)
<type> LOCATION(*)
INTEGER ADDRESS, IERR
```

Определение абсолютного байт-адреса **ADDRESS** размещения массива **LOCATION** в оперативной памяти компьютера. Адрес отсчитывается от базового адреса, значение которого содержится в системной константе **MPI_BOTTOM**. Процедура позволяет определять абсолютные адреса объектов как в языке Фортран, так и в Си, хотя в Си для этого предусмотрены иные средства. В языке Си параметр **ADDRESS** имеет тип **MPI_Aint**.

В следующем примере описывается новый тип данных **newtype**, который после регистрации используется для пересылки как единого целого двух элементов данных типов **double precision** и **character(1)**. В качестве адреса буфера посылки используется базовый адрес **MPI_BOTTOM**, а для определения смещений элементов данных **dat1** и **dat2** используются вызовы процедуры

MPI_ADDRESS. Перед пересылкой новый тип регистрируется при помощи вызова процедуры **MPI_TYPE_COMMIT**. Заметим, что пересылается один элемент данных производного типа, хотя он и состоит из двух разнотипных элементов.

```
      blocklens(1) = 1
      blocklens(2) = 1
      types(1) = MPI_DOUBLE_PRECISION
      types(2) = MPI_CHARACTER
      call MPI_ADDRESS(dat1, address(1), ierr)
      displs(1) = address(1)
      call MPI_ADDRESS(dat2, address(2), ierr)
      displs(2) = address(2)
      call MPI_TYPE_STRUCT(2, blocklens, displs, types,
     &                     newtype, ierr)
      call MPI_TYPE_COMMIT(newtype, ierr)
      call MPI_SEND(MPI_BOTTOM, 1, newtype, dest, tag,
     &              MPI_COMM_WORLD, ierr)
```

```
MPI_TYPE_LB(DATATYPE, DISPL, IERR)
INTEGER DATATYPE, DISPL, IERR
```

Определение смещения **DISPL** в байтах нижней границы элемента типа данных **DATATYPE** от начала буфера данных.

```
MPI_TYPE_UB(DATATYPE, DISPL, IERR)
INTEGER DATATYPE, DISPL, IERR
```

Определение смещения **DISPL** в байтах верхней границы элемента типа данных **DATATYPE** от начала буфера данных.

```
MPI_TYPE_EXTENT(DATATYPE, EXTENT, IERR)
INTEGER DATATYPE, EXTENT, IERR
```

Определение диапазона **EXTENT** (разницы между верхней и нижней границами элемента данного типа) типа данных **DATATYPE** в байтах.

В следующем примере производный тип данных используется для перестановки столбцов матрицы в обратном порядке. Тип данных **matr_rev**, создаваемый процедурой **MPI_TYPE_VECTOR**, описывает локальную часть матрицы данного процесса в переставленными в обратном порядке столбцами. После регистрации этот тип данных может использоваться при пересылке. Программа работает правильно, если размер матрицы **N** делится нацело на число процессов приложения.

```
 program example19
 include 'mpif.h'
 integer ierr, rank, size, N, nl
 parameter (N = 8)
 double precision a(N, N), b(N, N)
 call MPI_INIT(ierr)
 call MPI_COMM_SIZE(MPI_COMM_WORLD, size, ierr)
 call MPI_COMM_RANK(MPI_COMM_WORLD, rank, ierr)
 nl = (N-1)/size+1
 call work(a, b, N, nl, size, rank)
 call MPI_FINALIZE(ierr)
 end

 subroutine work(a, b, n, nl, size, rank)
 include 'mpif.h'
 integer ierr, rank, size, n, nl, ii, matr_rev
 double precision a(n, nl), b(n, nl)
 integer i, j, status(MPI_STATUS_SIZE)
 do j = 1, nl
    do i = 1, n
       b(i,j) = 0.d0
       ii = j+rank*nl
       a(i,j) = 100*ii+i
    enddo
 enddo
 call MPI_TYPE_VECTOR(nl, n, -n, MPI_DOUBLE_PRECISION,
&                     matr_rev, ierr)
 call MPI_TYPE_COMMIT(matr_rev, ierr)
 call MPI_SENDRECV(a(1, nl), 1, MATR_REV, size-rank-1, 1,
&                  b, nl*n, MPI_DOUBLE_PRECISION,
&                  size-rank-1, 1, MPI_COMM_WORLD,
&                  status, ierr)
 do j = 1, nl
    do i = 1, n
       print *, 'process ', rank, ': ',
&               j+rank*nl, ' ', i, a(i,j), ' ', b(i,j)
    enddo
 enddo
 end
```

Упаковка данных

Для пересылок разнородных данных наряду с созданием производных типов можно использовать операции упаковки и распаковки данных. Разнородные или расположенные не в последовательных ячейках памяти данные помещаются в один непрерывный буфер, пересылается, а потом полученное сообщение снова распределяется по нужным ячейкам памяти.

```
MPI_PACK(INBUF, INCOUNT, DATATYPE, OUTBUF, OUTSIZE, POSITION,
COMM, IERR)
<type> INBUF(*), OUTBUF(*)
INTEGER INCOUNT, DATATYPE, OUTSIZE, POSITION, COMM, IERR
```

Упаковка **INCOUNT** элементов типа **DATATYPE** из массива **INBUF** в массив **OUTBUF** со сдвигом **POSITION** байт от начала массива. Буфер **OUTBUF** должен содержать по крайней мере **OUTSIZE** байт. После выполнения процедуры 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_UNPACK(INBUF, INSIZE, POSITION, OUTBUF, OUTCOUNT, DATATYPE,
COMM, IERR)
<type> INBUF(*), OUTBUF(*)
INTEGER INSIZE, POSITION, OUTCOUNT, DATATYPE, COMM, IERR
```

Распаковка **OUTCOUNT** элементов типа **DATATYPE** из массива **INBUF** со сдвигом **POSITION** байт от начала массива в массив **OUTBUF**. Массив **INBUF** имеет размер не менее **INSIZE** байт.

```
MPI_PACK_SIZE(INCOUNT, DATATYPE, COMM, SIZE, IERR)
INTEGER INCOUNT, DATATYPE, COMM, SIZE, IERR
```

Определение необходимого объема памяти **SIZE** (в байтах) для упаковки **INCOUNT** элементов типа **DATATYPE**. Необходимый для упаковки размер может превышать сумму размеров пакуемых элементов данных.

В следующем примере массив **buf** используется в качестве буфера для упаковки **10** элементов массива **a** типа **real** и **10** элементов массива **b** типа **character**. Полученное сообщение пересылается процедурой **MPI_BCAST** от процесса **0** всем остальным процессам, полученное сообщение распаковывается при помощи вызовов процедур **MPI_UNPACK**.

```
      program example20
      include 'mpif.h'
      integer ierr, rank, position
      real a(10)
      character b(10), buf(100)
      call MPI_INIT(ierr)
      call MPI_COMM_RANK(MPI_COMM_WORLD, rank, ierr)
      do i = 1, 10
         a(i) = rank + 1.0
         if(rank .eq. 0) then
            b(i) = 'a'
         else
            b(i) = 'b'
         end if
      end do
      position=0
```

```
 if(rank .eq. 0) then
    call MPI_PACK(a, 10, MPI_REAL, buf, 100, position,
&                 MPI_COMM_WORLD, ierr)
    call MPI_PACK(b, 10, MPI_CHARACTER, buf, 100, position,
&                 MPI_COMM_WORLD, ierr)
    call MPI_BCAST(buf, 100, MPI_PACKED, 0,
&                  MPI_COMM_WORLD, ierr)
 else
    call MPI_BCAST(buf, 100, MPI_PACKED, 0,
&                  MPI_COMM_WORLD, ierr)
    position=0
    call MPI_UNPACK(buf, 100, position, a, 10, MPI_REAL,
&                   MPI_COMM_WORLD, ierr)
    call MPI_UNPACK(buf, 100, position, b, 10,
&                   MPI_CHARACTER, MPI_COMM_WORLD, ierr)
 end if
 print *, 'procecc ', rank, ' a=', a, ' b=', b
 call MPI_FINALIZE(ierr)
 end
```

Задания

- Создать производный тип данных для пересылок диагональной матрицы.
- Как соотносятся значения, возвращаемые процедурами **MPI_TYPE_SIZE** и **MPI_TYPE_EXTENT**?
- Можно ли использовать производные типы данных без вызова процедуры **MPI_TYPE_COMMIT**?
- Переслать нулевому процессу от всех процессов приложения структуру, состоящую из ранга процесса и названия узла, на котором данный процесс запущен (полученного с помощью процедуры **MPI_GET_PROCESSOR_NAME**).
- Прямоугольная матрица распределена по процессам по строкам. Переставить строки матрицы в обратном порядке, используя для пересылок производный тип данных.
- Сделать предыдущую задачу с использованием пересылок упакованных данных.
- В чем преимущества и недостатки использования пересылок упакованных данных по сравнению с пересылками данных производных типов?
- Написать программу транспонирования матрицы с использованием производных типов данных.