```
call MPI_FINALIZE(ierr)
end
```

Задания

- Какие группы процессов существуют при запуске приложения?
- Могут ли группы процессов иметь непустое пересечение, не совпадающее ни с одной из них полностью?
- В чем отличие между группой процессов и коммуникатором?
- Могут ли обмениваться данными процессы, принадлежащие разным коммуникаторам?
- Может ли в какой-то группе не быть процесса с номером **0**?
- Может ли в какую-либо группу не войти процесс с номером **0** в коммуникаторе **MPI_COMM_WORLD**?
- Может ли только один процесс в некоторой группе вызвать процедуру **MPI_GROUP_INCL**?
- Как создать новую группу из процессов **3**, **4** и **7** коммуникатора **MPI_COMM_WORLD**?
- Разбить все процессы приложения на три произвольных группы и напечатать ранги в **MPI_COMM_WORLD** тех процессов, что попали в первые две группы, но не попали в третью.
- Какие коммуникаторы существуют при запуске приложения?
- Можно ли в процессе выполнения программы изменить число процессов в коммуникаторе **MPI_COMM_WORLD**?
- Может ли только один процесс в некотором коммуникаторе вызвать процедуру **MPI_COMM_CREATE**?
- Можно ли при помощи процедуры **MPI_COMM_SPLIT** создать ровно один новый коммуникатор?
- Можно ли при помощи процедуры **MPI_COMM_SPLIT** создать столько новых коммуникаторов, сколько процессов входит в базовый коммуникатор?
- Реализовать разбиение процессов на две группы, в одной из которых осуществляется обмен данными по кольцу, а в другой – коммуникации по схеме master-slave.

Виртуальные топологии

Топология – это механизм сопоставления процессам некоторого коммуникатора альтернативной схемы адресации. В MPI топологии виртуальны, то есть они не связаны с физической топологией коммуникационной сети. Тополо-

гия используется программистом для более удобного обозначения процессов, и таким образом, приближения параллельной программы к структуре математического алгоритма. Кроме того, топология может использоваться системой для оптимизации распределения процессов по физическим процессорам используемого параллельного компьютера при помощи изменения порядка нумерации процессов внутри коммуникатора.

В MPI предусмотрены два типа топологий:

- *декартова топология* (прямоугольная решетка произвольной размерности);
- *топология графа*.

```
MPI_TOPO_TEST(COMM, TYPE, IERR)
INTEGER COMM, TYPE, IERR
```

Процедура определения типа топологии, связанной с коммуникатором **COMM**. Возможные возвращаемые значения параметра **TYPE**:

- **MPI_GRAPH** для топологии графа;
- **MPI_CART** для декартовой топологии;
- **MPI_UNDEFINED** – с коммуникатором **COMM** не связана никакая топология.

Декартова топология

```
MPI_CART_CREATE(COMM, NDIMS, DIMS, PERIODS, REORDER, COMM_CART,
IERR)
INTEGER COMM, NDIMS, DIMS(*), COMM_CART, IERR
LOGICAL PERIODS(*), REORDER
```

Создание коммуникатора **COMM_CART**, обладающего декартовой топологией, из процессов коммуникатора **COMM**. Параметр **NDIMS** задает размерность получаемой декартовой решетки, **DIMS(I)** – число элементов в измерении **I**, **1≤I≤NDIMS**. **PERIODS** – логический массив из **NDIMS** элементов, определяющий, является ли решетка периодической (значение **.TRUE.**) вдоль каждого измерения. **REORDER** – логический параметр, определяющий, что при значении **.TRUE.** системе разрешено менять порядок нумерации процессов для оптимизации распределения процессов по физическим процессорам используемого параллельного компьютера.

Процедура является коллективной, а значит, должна быть вызвана всеми процессами коммуникатора **COMM**. Если количество процессов в задаваемой топологии **COMM_CART** меньше числа процессов в исходном коммуникаторе **COMM**, то некоторым процессам может вернуться значение **MPI_COMM_NULL**, а значит, они не будут принимать участия в создаваемой топологии. Если ко-

личество процессов в задаваемой топологии больше числа процессов в исходном коммуникаторе, то вызов будет ошибочным.

В следующем примере создается трехмерная топология **4×3×2**, каждое измерение которой является периодическим, кроме того, разрешается переупорядочение процессов. Данный фрагмент должен выполняться не менее чем на **24** процессах.

```
      dims(1) = 4
      dims(2) = 3
      dims(3) = 2
      periods(1) = .TRUE.
      periods(2) = .TRUE.
      periods(3) = .TRUE.
      call MPI_CART_CREATE(MPI_COMM_WORLD, 3, dims, periods,
     &                     .TRUE., comm_cart, ierr)
```

```
MPI_DIMS_CREATE(NNODES, NDIMS, DIMS, IERR)
INTEGER NNODES, NDIMS, DIMS(*), IERR
```

Процедура помогает определить размеры **DIMS(I)** для каждой из **NDIMS** размерностей при создании декартовой топологии для **NNODES** процессов. Предпочтительным считается создание топологии, в которой число процессов по разным размерностям примерно одно и то же. Пользователь может управлять числом процессов в некоторых размерностях следующим образом. Значение **DIMS(I)** рассчитывается данной процедурой, если перед вызовом оно равно **0**, иначе оставляется без изменений. Отрицательные значения элементов массива **DIMS** являются ошибочными. Перед вызовом процедуры значение **NNODES** должно быть кратно произведению ненулевых значений массива **DIMS**. Выходные значения массива **DIMS**, переопределенные данной процедурой, будут упорядочены в порядке убывания. Процедура является локальной и не требует межпроцессного взаимодействия.

В следующей таблице приведены четыре примера использования процедуры **MPI_DIMS_CREATE** для создания трехмерных топологий. В первом примере **6** процессов образуют решетку **3×2×1**, причем размеры упорядочены в порядке убывания. Во втором примере делается попытка распределить **7** процессов по трем измерениям, единственный возможный вариант – решетка **7×1×1**. В третьем примере для второй размерности изначально задано значение **3**, две оставшиеся размерности определяют решетку **2×3×1**. Четвертый вызов ошибочен, так как общее число процессов (**7**) не делится нацело на заданный размер во второй размерности (**3**).

dims перед вызовом	вызов процедуры	**dims** после вызова
(0, 0, 0)	MPI_DIMS_CREATE(6, 3, dims, ierr)	(3, 2, 1)
(0, 0, 0)	MPI_DIMS_CREATE(7, 3, dims, ierr)	(7, 1, 1)
(0, 3, 0)	MPI_DIMS_CREATE(6, 3, dims, ierr)	(2, 3, 1)
(0, 3, 0)	MPI_DIMS_CREATE(7, 3, dims, ierr)	ошибка

```
MPI_CART_COORDS(COMM, RANK, MAXDIMS, COORDS, IERR)
INTEGER COMM, RANK, MAXDIMS, COORDS(*), IERR
```

Определение декартовых координат процесса по его рангу **RANK** в коммуникаторе **COMM**. Координаты возвращаются в массиве **COORDS** с числом элементов **MAXDIMS**. Отсчет координат по каждому измерению начинается с нуля.

```
MPI_CART_RANK(COMM, COORDS, RANK, IERR)
INTEGER COMM, COORDS(*), RANK, IERR
```

Определение ранга **RANK** процесса в коммуникаторе **COMM** по его декартовым координатам **COORDS**. Для периодических решеток координаты вне допустимых интервалов пересчитываются, для непериодических решеток они являются ошибочными.

```
MPI_CART_SUB(COMM, DIMS, NEWCOMM, IERR)
INTEGER COMM, NEWCOMM, IERR
LOGICAL DIMS(*)
```

Расщепление коммуникатора **COMM**, с которым связана декартова топология при помощи процедуры **MPI_CART_CREATE**, на подгруппы, соответствующие декартовым подрешеткам меньшей размерности. **I**–ый элемент логического массива **DIMS** устанавливается равным значению **.TRUE.**, если **I**–ое измерение должно остаться в формируемой подрешетке, связанной с коммуникатором **NEWCOMM**.

Возьмем трехмерную топологию, созданную в предыдущем примере. Ниже показано, как расщепить топологию **4×3×2** на **3** двумерных подрешетки **4×2** по **8** процессов в каждой.

```
      dims(0) = .TRUE.
      dims(1) = .FALSE.
      dims(2) = .TRUE.
      call MPI_CART_SUB(comm_cart, dims, newcomm, ierr)
```

```
MPI_CARTDIM_GET(COMM, NDIMS, IERR)
INTEGER COMM, NDIMS, IERR
```

Определение размерности **NDIMS** декартовой топологии, связанной с коммуникатором **COMM**.

```
MPI_CART_GET(COMM, MAXDIMS, DIMS, PERIODS, COORDS, IERR)
INTEGER COMM, MAXDIMS, DIMS(*), COORDS(*), IERR
```

LOGICAL PERIODS(*)

Получение информации о декартовой топологии коммуникатора **COMM** и координатах в ней вызвавшего процесса. **MAXDIMS** задает размерность декартовой топологии. В параметре **DIMS** возвращается количество процессов для каждого измерения, в параметре **PERIODS** – периодичность по каждому измерению, в параметре **COORDS** – координаты вызвавшего процесса в декартовой топологии.

MPI_CART_SHIFT(COMM, DIRECTION, DISP, SOURCE, DEST, IERR)
INTEGER COMM, DIRECTION, DISP, SOURCE, DEST, IERR

Получение номеров посылающего (**SOURCE**) и принимающего (**DEST**) процессов в декартовой топологии коммуникатора **COMM** для осуществления сдвига вдоль измерения **DIRECTION** на величину **DISP**.

Для периодических измерений осуществляется циклический сдвиг, для непериодических – линейный сдвиг. В случае линейного сдвига на некоторых процессах в качестве номеров посылающего или принимающего процессов может быть получено значение **MPI_PROC_NULL**, означающее выход за границы диапазона. В случае циклического сдвига последний процесс по данному измерению осуществляет обмены с нулевым процессом. Для **n**-мерной декартовой решетки значение **DIRECTION** должно быть в пределах от **0** до **n-1**.

Значения **SOURCE** и **DEST** можно использовать, например, для обмена с помощью процедуры **MPI_SENDRECV**.

В следующем примере создается двумерная декартова решетка, периодическая по обоим измерениям, определяются координаты процесса в данной решетке. Потом при помощи процедуры **MPI_CART_SHIFT** вычисляются координаты процессов, с которыми нужно совершить обмен данными для осуществления циклического сдвига с шагом **2** по измерению **1**. В конце фрагмента полученные значения номеров процессов используются для обмена данными при помощи процедуры **MPI_SENDRECV_REPLACE**.

```
      periods(1) = .TRUE.
      periods(2) = .TRUE.
      call MPI_CART_CREATE(MPI_COMM_WORLD, 2, dims,
     &                     periods, .TRUE., comm, ierr)
      call MPI_COMM_RANK(comm, rank, ierr)
      call MPI_CART_COORDS(comm, rank, 2, coords, ierr)
      shift = 2
      dest = 1
      call MPI_CART_SHIFT(comm, 0, shift, source, dest, ierr)
      call MPI_SENDRECV_REPLACE(a, 1, MPI_REAL, dest, 0,
     &                          source, 0, comm, status, ierr)
```

Топология графа

```
MPI_GRAPH_CREATE(COMM, NNODES, INDEX, EDGES, REORDER,
COMM_GRAPH, IERR)
INTEGER COMM, NNODES, INDEX(*), EDGES(*), COMM_GRAPH, IERR
LOGICAL REORDER
```

Создание на основе коммуникатора **COMM** нового коммуникатора **COMM_GRAPH** с топологией графа. Параметр **NNODES** задает число вершин графа, **INDEX(I)** содержит суммарное количество соседей для первых **I** вершин. Массив **EDGES** содержит упорядоченный список номеров процессов-соседей всех вершин. Параметр **REORDER** при значении **.TRUE.** означает, что системе разрешено менять порядок нумерации процессов.

Процедура является коллективной, а значит, должна быть вызвана всеми процессами исходного коммуникатора. Если **NNODES** меньше числа процессов коммуникатора **COMM**, то некоторым процессам вернется значение **MPI_COMM_NULL**, а значит, они не будут принимать участия в создаваемой топологии. Если **NNODES** больше числа процессов коммуникатора **COMM**, то вызов процедуры является ошибочным.

В следующей табличке приведен пример описания графа через задание всех соседей каждой вершины.

Процесс	Соседи
0	1, 3
1	0
2	3
3	0, 2

Для описания такого графа нужно заполнить следующие структуры данных:

```
INDEX=2, 3, 4, 6
EDGES=1, 3, 0, 3, 0, 2
```

После этого можно создать топологию графа, например, с помощью следующего вызова (вызов будет корректным при выполнении на не менее чем на **4** процессах):

```
      call MPI_GRAPH_CREATE(MPI_COMM_WORLD, 4, INDEX, EDGES,
     &                      .TRUE., comm_graph, ierr)
```

```
MPI_GRAPH_NEIGHBORS_COUNT(COMM, RANK, NNEIGHBORS, IERR)
INTEGER COMM, RANK, NNEIGHBORS, IERR
```
Определение количества **NNEIGHBORS** непосредственных соседей процесса с рангом **RANK** в графовой топологии, связанной с коммуникатором **COMM**.

```
MPI_GRAPH_NEIGHBORS(COMM, RANK, MAX, NEIGHBORS, IERR)
INTEGER COMM, RANK, MAX, NEIGHBORS(*), IERR
```
Определение рангов непосредственных соседей процесса с рангом **RANK** в графовой топологии, связанной с коммуникатором **COMM**. Ранги соседей возвращаются в массиве **NEIGHBORS**, **MAX** задает ограничение на количество соседей (может быть получено, например, вызовом процедуры **MPI_GRAPH_NEIGHBORS_COUNT**).

```
MPI_GRAPHDIMS_GET(COMM, NNODES, NEDGES, IERR)
INTEGER COMM, NNODES, NEDGES, IERR
```
Определение числа вершин **NNODES** и числа ребер **NEDGES** графовой топологии, связанной с коммуникатором **COMM**.

```
MPI_GRAPH_GET(COMM, MAXINDEX, MAXEDGES, INDEX, EDGES, IERR)
INTEGER COMM, MAXINDEX, MAXEDGES, INDEX(*), EDGES(*), IERR
```
Определение информации о топологии графа, связанной с коммуникатором **COMM**. В массивах **INDEX** и **EDGES** возвращается описание графовой топологии в том виде, как она задается при создании топологии с помощью процедуры **MPI_GRAPH_CREATE**. Параметры **MAXINDEX** и **MAXEDGES** задают ограничения на размеры соответствующих массивов (могут быть получены, например, вызовом процедуры **MPI_GRAPHDIMS_GET**).

В следующем примере создается графовая топология **comm_graph** для общения процессов по коммуникационной схеме master-slave. Все процессы в рамках данной топологии могут общаться только с нулевым процессом. После создания топологии с помощью вызова процедуры **MPI_GRAPH_CREATE** каждый процесс определяет количество своих непосредственных соседей в рамках данной топологии (с помощью вызова процедуры **MPI_GRAPH_NEIGHBORS_COUNT**) и ранги процессов-соседей (с помощью вызова процедуры **MPI_GRAPH_NEIGHBORS**). После этого каждый процесс может в рамках данной топологии обмениваться данными со своими непосредственными соседями, например, при помощи вызова процедуры **MPI_SENDRECV**.

```
 program example18
 include 'mpif.h'
 integer ierr, rank, rank1, i, size, MAXPROC, MAXEDGES
 parameter (MAXPROC = 128, MAXEDGES = 512)
 integer a, b
 integer status(MPI_STATUS_SIZE)
 integer comm_graph, index(MAXPROC), edges(MAXEDGES)
 integer num, neighbors(MAXPROC)
 call MPI_INIT(ierr)
 call MPI_COMM_SIZE(MPI_COMM_WORLD, size, ierr)
 call MPI_COMM_RANK(MPI_COMM_WORLD, rank, ierr)
 do i = 1, size
    index(i) = size+i-2
 end do
 do i = 1, size-1
    edges(i) = i
    edges(size+i-1) = 0
 end do
 call MPI_GRAPH_CREATE(MPI_COMM_WORLD, size, index, edges,
&                      .TRUE., comm_graph, ierr)
 call MPI_GRAPH_NEIGHBORS_COUNT(comm_graph, rank, num,
&                               ierr)
 call MPI_GRAPH_NEIGHBORS(comm_graph, rank, num, neighbors,
&                         ierr)
 do i = 1, num
    call MPI_SENDRECV(rank, 1, MPI_INTEGER, neighbors(i),
&                     1, rank1, 1, MPI_INTEGER,
&                     neighbors(i), 1, comm_graph,
&                     status, ierr)
    print *, 'procecc ', rank, ' communicate with process',
&            rank1
 end do
 call MPI_FINALIZE(ierr)
 end
```

Задания

- Обязана ли виртуальная топология повторять физическую топологию целевого компьютера?
- Любой ли коммуникатор может обладать виртуальной топологией?
- Может ли процесс входить одновременно в декартову топологию и в топологию графа?
- Можно ли вызвать процедуру **MPI_CART_CREATE** только на половине процессов коммуникатора?
- Необходимо ли использовать процедуру **MPI_DIMS_CREATE** перед вызовом процедуры **MPI_CART_CREATE**?
- Можно ли использовать координаты процесса в декартовой топологии в процедурах обмена данными?

- 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.