- Смоделировать глобальное суммирование методом сдваивания и сравнить эффективность такой реализации с использованием стандартной процедуры **MPI_REDUCE**.
- Смоделировать процедуру **MPI_ALLREDUCE** при помощи процедур **MPI_REDUCE** и **MPI_BCAST**.
- Напишите свой вариант процедуры **MPI_GATHER**, используя функции посылки сообщений типа точка-точка.
- Подумайте, как организовать коллективный асинхронный обмен данными, аналогичный функции: а) **MPI_REDUCE**; б) **MPI_ALLTOALL**.
- Исследовать масштабируемость (зависимость времени выполнения от числа процессов) различных коллективных операций на конкретной системе.

Группы и коммуникаторы

В MPI существуют широкие возможности для операций над группами процессов и коммуникаторами. Это бывает необходимо, во-первых, чтобы дать возможность некоторой группе процессов работать над своей независимой подзадачей. Во-вторых, если особенность алгоритма такова, что только часть процессов должна обмениваться данными, бывает удобно завести для их взаимодействия отдельный коммуникатор. В-третьих, при создании библиотек подпрограмм нужно гарантировать, что пересылки данных в библиотечных модулях не пересекутся с пересылками в основной программе. Решение этих задач можно обеспечить в полном объеме только при помощи создания нового независимого коммуникатора.

Операции с группами процессов

Группа – это упорядоченное множество процессов. Каждому процессу в группе сопоставлено целое число – *ранг* или *номер*. **MPI_GROUP_EMPTY** – пустая группа, не содержащая ни одного процесса. **MPI_GROUP_NULL** – значение, используемое для ошибочной группы.

Новые группы можно создавать как на основе уже существующих групп, так и на основе коммуникаторов, но в операциях обмена могут использоваться только коммуникаторы. Базовая группа, из которой создаются все остальные группы процессов, связана с коммуникатором **MPI_COMM_WORLD**, в нее входят все процессы приложения. Операции над группами процессов являются локальными, в них вовлекается только вызвавший процедуру процесс, а выполнение не требует межпроцессного обмена данными. Любой процесс может производить операции над любыми группами, в том числе над такими,

которые не содержат данный процесс. При операциях над группами может получиться пустая группа **MPI_GROUP_EMPTY**.

```
MPI_COMM_GROUP(COMM, GROUP, IERR)
INTEGER COMM, GROUP, IERR
```

Получение группы **GROUP**, соответствующей коммуникатору **COMM**. В языке Си параметр **GROUP** имеет предопределенный тип **MPI_Group**. Поскольку изначально существует единственный нетривиальный коммуникатор **MPI_COMM_WORLD**, сначала нужно получить соответствующую ему группу процессов. Это можно сделать при помощи следующего вызова:

```
call MPI_COMM_GROUP(MPI_COMM_WORLD, group, ierr)
```

```
MPI_GROUP_INCL(GROUP, N, RANKS, NEWGROUP, IERR)
INTEGER GROUP, N, RANKS(*), NEWGROUP, IERR
```

Создание группы **NEWGROUP** из **N** процессов прежней группы **GROUP** с рангами **RANKS(1),…,RANKS(N)**, причем рангу **RANKS(I)** в старой группе соответствует ранг **I-1** в новой группе. При **N=0** создается пустая группа **MPI_GROUP_EMPTY**. Возможно использование этой процедуры для задания нового порядка процессов в группе.

```
MPI_GROUP_EXCL(GROUP, N, RANKS, NEWGROUP, IERR)
INTEGER GROUP, N, RANKS(*), NEWGROUP, IERR
```

Создание группы **NEWGROUP** из процессов группы **GROUP**, исключая процессы с рангами **RANKS(1),…,RANKS(N)**, причем порядок оставшихся процессов в новой группе соответствует порядку процессов в старой группе. При **N=0** создается группа, идентичная старой группе.

В следующем примере создается две непересекающихся группы процессов **group1** и **group2** на основе процессов группы **group**. В каждую из создаваемых групп войдет примерно половина процессов прежней группы (при нечетном числе процессов в группу **group2** войдет на один процесс больше). Порядок нумерации процессов во вновь создаваемых группах сохранится.

```
size1 = size/2
do i = 1, size1
   ranks(i) = i-1
enddo
call MPI_GROUP_INCL(group, size1, ranks, group1, ierr)
call MPI_GROUP_EXCL(group, size1, ranks, group2, ierr)
```

Следующие три процедуры определяют операции над группами процессов, как над множествами. Из-за особенностей нумерации процессов ни объединение, ни пересечение групп не коммутативны, но ассоциативны.

```
MPI_GROUP_INTERSECTION(GROUP1, GROUP2, NEWGROUP, IERR)
INTEGER GROUP1, GROUP2, NEWGROUP, IERR
```

Создание группы **NEWGROUP** из пересечения групп **GROUP1** и **GROUP2**. Полученная группа содержит все процессы группы **GROUP1**, входящие также в группу **GROUP2** и упорядоченные, как в первой группе.

```
MPI_GROUP_UNION(GROUP1, GROUP2, NEWGROUP, IERR)
INTEGER GROUP1, GROUP2, NEWGROUP, IERR
```

Создание группы **NEWGROUP** из объединения групп **GROUP1** и **GROUP2**. Полученная группа содержит все процессы группы **GROUP1** в прежнем порядке, за которыми следуют процессы группы **GROUP2**, не вошедшие в группу **GROUP1**, также в прежнем порядке.

```
MPI_GROUP_DIFFERENCE(GROUP1, GROUP2, NEWGROUP, IERR)
INTEGER GROUP1, GROUP2, NEWGROUP, IERR
```

Создание группы **NEWGROUP** из разности групп **GROUP1** и **GROUP2**. Полученная группа содержит все элементы группы **GROUP1**, не входящие в группу **GROUP2** и упорядоченные, как в первой группе.

Например, пусть в группу **gr1** входят процессы **0**, **1**, **2**, **4**, **5**, а в группу **gr2** - процессы **0**, **2**, **3** (нумерация процессов задана в группе, соответствующей коммуникатору **MPI_COMM_WORLD**). Тогда после вызовов

```
call MPI_GROUP_INTERSECTION(gr1, gr2, newgr1, ierr)
call MPI_GROUP_UNION(gr1, gr2, newgr2, ierr)
call MPI_GROUP_DIFFERENCE(gr1, gr2, newgr3, ierr)
```

в группу **newgr1** входят процессы **0**, **2**;
в группу **newgr2** входят процессы **0**, **1**, **2**, **4**, **5**, **3**;
в группу **newgr3** входят процессы **1**, **4**, **5**.

Порядок нумерации процессов в полученных группах соответствует порядку их перечисления.

```
MPI_GROUP_SIZE(GROUP, SIZE, IERR)
INTEGER GROUP, SIZE, IERR
```

Определение количества **SIZE** процессов в группе **GROUP**.

```
MPI_GROUP_RANK(GROUP, RANK, IERR)
INTEGER GROUP, RANK, IERR
```

Определение номера процесса **RANK** в группе **GROUP**. Если вызвавший процесс не входит в группу **GROUP**, то возвращается значение **MPI_UNDEFINED**.

```
MPI_GROUP_TRANSLATE_RANKS(GROUP1, N, RANKS1, GROUP2, RANKS2,
IERR)
INTEGER GROUP1, N, RANKS1(*), GROUP2, RANKS2(*), IERR
```

В массиве **RANKS2** возвращаются ранги в группе **GROUP2** процессов с рангами **RANKS1** в группе **GROUP1**. Параметр **N** задает число процессов, для которых нужно определить ранги.

```
MPI_GROUP_COMPARE(GROUP1, GROUP2, RESULT, IERR)
INTEGER GROUP1, GROUP2, RESULT, IERR
```

Сравнение групп **GROUP1** и **GROUP2**. Если группы **GROUP1** и **GROUP2** полностью совпадают, то в параметре **RESULT** возвращается значение **MPI_IDENT**. Если группы отличаются только рангами процессов, то возвращается значение **MPI_SIMILAR**. Иначе возвращается значение **MPI_UNEQUAL**.

```
MPI_GROUP_FREE(GROUP, IERR)
INTEGER GROUP, IERR
```

Уничтожение группы **GROUP**. После выполнения процедуры переменная **GROUP** принимает значение **MPI_GROUP_NULL**. Если с этой группой к моменту вызова процедуры уже выполняется какая-то операция, то она будет завершена.

В следующем примере все процессы приложения разбиваются на две непересекающиеся примерно равные группы **group1** и **group2**. При нечетном числе процессов в группе **group2** может оказаться на один процесс больше, тогда последний процесс из данной группы не должен обмениваться данными ни с одним процессом из группы **group1**. С помощью вызовов процедуры **MPI_GROUP_TRANSLATE_RANKS** каждый процесс находит процесс с тем же номером в другой группе и обменивается с ним сообщением через коммуникатор **MPI_COMM_WORLD** при помощи вызова процедуры **MPI_SENDRECV**. В конце программы не нужные далее группы уничтожаются с помощью вызовов процедур **MPI_GROUP_FREE**.

```
      program example16
      include 'mpif.h'
      integer ierr, rank, i, size, size1
      integer a(4), b(4)
      integer status(MPI_STATUS_SIZE)
      integer group, group1, group2
      integer ranks(128), rank1, rank2, rank3
      call MPI_INIT(ierr)
      call MPI_COMM_SIZE(MPI_COMM_WORLD, size, ierr)
      call MPI_COMM_RANK(MPI_COMM_WORLD, rank, ierr)
      call MPI_COMM_GROUP(MPI_COMM_WORLD, group, ierr)
      size1 = size/2
      do i = 1, size1
         ranks(i) = i-1
      enddo
      call MPI_GROUP_INCL(group, size1, ranks, group1, ierr)
      call MPI_GROUP_EXCL(group, size1, ranks, group2, ierr)
      call MPI_GROUP_RANK(group1, rank1, ierr)
      call MPI_GROUP_RANK(group2, rank2, ierr)
```

```
 if (rank1 .eq. MPI_UNDEFINED) then
    if(rank2 .lt. size1) then
       call MPI_GROUP_TRANSLATE_RANKS(group1, 1, rank2,
&                                     group, rank3, ierr)
    else
       rank3 = MPI_UNDEFINED
    end if
 else
    call MPI_GROUP_TRANSLATE_RANKS(group2, 1, rank1,
&                                  group, rank3, ierr)
 end if
 a(1) = rank
 a(2) = rank1
 a(3) = rank2
 a(4) = rank3
 if (rank3 .ne. MPI_UNDEFINED) then
    call MPI_SENDRECV(a, 4, MPI_INTEGER, rank3, 1,
&                     b, 4, MPI_INTEGER, rank3, 1,
&                     MPI_COMM_WORLD, status, ierr)
 end if
 call MPI_GROUP_FREE(group, ierr)
 call MPI_GROUP_FREE(group1, ierr)
 call MPI_GROUP_FREE(group2, ierr)
 print *, 'process ', rank, ' a=', a, ' b=', b
 call MPI_FINALIZE(ierr)
 end
```

Операции с коммуникаторами

Коммуникатор предоставляет отдельный контекст обмена процессов некоторой группы. Контекст обеспечивает возможность независимых обменов данными. Каждой группе процессов может соответствовать несколько коммуникаторов, но каждый коммуникатор в любой момент времени однозначно соответствует только одной группе.

Следующие коммуникаторы создаются сразу после вызова процедуры **`MPI_INIT`**:

- **`MPI_COMM_WORLD`** – коммуникатор, объединяющий все процессы приложения;
- **`MPI_COMM_NULL`** – значение, используемое для ошибочного коммуникатора;
- **`MPI_COMM_SELF`** – коммуникатор, включающий только вызвавший процесс.

Создание коммуникатора является коллективной операцией и требует операции межпроцессного обмена, поэтому такие процедуры должны вызываться всеми процессами некоторого существующего коммуникатора.

```
MPI_COMM_DUP(COMM, NEWCOMM, IERR)
INTEGER COMM, NEWCOMM, IERR
```

Создание нового коммуникатора **NEWCOMM** с той же группой процессов и атрибутами, что и у коммуникатора **COMM**.

```
MPI_COMM_CREATE(COMM, GROUP, NEWCOMM, IERR)
INTEGER COMM, GROUP, NEWCOMM, IERR
```

Создание нового коммуникатора **NEWCOMM** из коммуникатора **COMM** для группы процессов **GROUP**, которая должна являться подмножеством группы, связанной с коммуникатором **COMM**. Вызов должен встретиться во всех процессах коммуникатора **COMM**. На процессах, не принадлежащих группе **GROUP**, будет возвращено значение **MPI_COMM_NULL**.

В следующем примере создается две новых группы, одна из которых содержит первую половину процессов, а вторая – вторую половину. При нечетном числе процессов во вторую группу войдет на один процесс больше. Каждая группа создается только на тех процессах, которые в нее входят. Для каждой новой группы создается соответствующий ей коммуникатор **new_comm**, и операция **MPI_ALLREDUCE** выполняется по отдельности для процессов, входящих в разные группы.

```
 call MPI_COMM_GROUP(MPI_COMM_WORLD, group, ierr)
 do i = 1, size/2
    ranks(i) = i-1
 end do
 if (rank .lt. size/2) then
    call MPI_GROUP_INCL(group, size/2, ranks,
&                        new_group, ierr)
 else
    call MPI_GROUP_EXCL(group, size/2, ranks,
&                        new_group, ierr)
 end if
 call MPI_COMM_CREATE(MPI_COMM_WORLD, new_group,
&                     new_comm, ierr)
 call MPI_ALLREDUCE(sbuf, rbuf, 1, MPI_INTEGER,
&                   MPI_SUM, new_comm, ierr)
 call MPI_GROUP_RANK(new_group, new_rank, ierr)
 print *, 'rank= ', rank, ' newrank= ',
&         new_rank, ' rbuf= ', rbuf
```

```
MPI_COMM_SPLIT(COMM, COLOR, KEY, NEWCOMM, IERR)
INTEGER COMM, COLOR, KEY, NEWCOMM, IERR
```
Разбиение коммуникатора **COMM** на несколько новых коммуникаторов по числу значений параметра **COLOR**. В один коммуникатор попадают процессы с одним значением **COLOR**. Процессы с б**о**льшим значением параметра **KEY** получат больший ранг в новой группе, при одинаковом значении параметра **KEY** порядок нумерации процессов выбирается системой.

Процессы, которые не должны войти в новые коммуникаторы, указывают в качестве параметра **COLOR** константу **MPI_UNDEFINED**. Им в параметре **NEWCOMM** вернется значение **MPI_COMM_NULL**.

В следующем примере коммуникатор **MPI_COMM_WORLD** разбивается на три части. В первую войдут процессы с номерами **0**, **3**, **6** и т.д., во вторую – **1**, **4**, **7** и т.д., а в третью – **2**, **5**, **8** и т.д. Задание в качестве параметра **KEY** переменной **rank** гарантирует, что порядок нумерации процессов в создаваемых группах соответствует порядку нумерации в исходной группе, то есть, порядку перечисления выше.

```
      call MPI_COMM_SPLIT(MPI_COMM_WORLD, mod(rank, 3),
     &                    rank, new_comm, ierr)
```

```
MPI_COMM_FREE(COMM, IERR)
INTEGER COMM, IERR
```
Удаление коммуникатора **COMM**. После выполнения процедуры переменной **COMM** присваивается значение **MPI_COMM_NULL**. Если с этим коммуникатором к моменту вызова процедуры уже выполняется какая-то операция, то она будет завершена.

В следующем примере создается один новый коммуникатор **comm_revs**, в который входят все процессы приложения, пронумерованные в обратном порядке. Когда коммуникатор становится ненужным, он удаляется при помощи вызова процедуры **MPI_COMM_FREE**. Так можно использовать процедуру **MPI_COMM_SPLIT** для перенумерации процессов.

```
      program example17
      include 'mpif.h'
      integer ierr, rank, size
      integer comm_revs, rank1
      call MPI_INIT(ierr)
      call MPI_COMM_SIZE(MPI_COMM_WORLD, size, ierr)
      call MPI_COMM_RANK(MPI_COMM_WORLD, rank, ierr)
      call MPI_COMM_SPLIT(MPI_COMM_WORLD, 1, size-rank,
     &                    comm_revs, ierr)
      call MPI_COMM_RANK(comm_revs, rank1, ierr)
      print *, 'rank = ', rank,' rank1 = ', rank1
      call MPI_COMM_FREE(comm_revs, ierr)
```

```
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-