5.6. Управление группами процессов и коммуникаторами

Рассмотрим теперь возможности MPI по управлению группами процессов и коммуникаторами.

Для изложения последующего материала напомним ряд понятий и определений, приведенных в начале данного раздела.

Процессы параллельной программы объединяются в **группы**. В группу могут входить все процессы параллельной программы; с другой стороны, в группе может находиться только часть имеющихся процессов. Соответственно, один и тот же процесс может принадлежать нескольким группам. Управление группами процессов предпринимается для создания на их основе коммуникаторов.

Под **коммуникатором** в MPI понимается специально создаваемый служебный объект, объединяющий в своем составе группу процессов и ряд дополнительных параметров (**контекст**), используемых при выполнении операций передачи данных. Как правило, парные операции передачи данных выполняются для процессов, принадлежащих одному и тому же коммуникатору. Коллективные операции применяются одновременно для всех процессов коммуникатора. Создание коммуникаторов предпринимается для уменьшения области действия коллективных операций и для устранения взаимовлияния разных выполняемых частей параллельной программы. Важно еще раз подчеркнуть – коммуникационные операции, выполняемые с использованием разных коммуникаторов, являются независимыми и не влияют друг на друга.

Все имеющиеся в параллельной программе процессы входят в состав создаваемого по умолчанию коммуникатора с идентификатором MPI_COMM_WORLD.

При необходимости передачи данных между процессами из разных групп необходимо создавать глобальный коммуникатор (**intercommunicator**). Взаимодействие между процессами разных групп оказывается необходимым в достаточно редких ситуациях, в данном учебном материале не рассматривается и может служить темой для самостоятельно изучения – см., например, Немнюгин и Стесик (2002), Group, et al. (1994), Pacheco (1996).

5.6.1. Управление группами

Группы процессов могут быть созданы только из уже существующих групп. В качестве исходной группы может быть использована группа, связанная с предопределенным коммуникатором MPI_COMM_WORLD.

Для получения группы, связанной с существующим коммуникатором, используется функция:

```
int MPI_Comm_group( MPI_Comm comm, MPI_Group *group ).
```

Далее, на основе существующих групп, могут быть созданы новые группы:

- создание новой группы **newgroup** из существующей группы **oldgroup**, которая будет включать в себя n процессов, ранги которых перечисляются в массиве **ranks**:

  ```
  int MPI_Group_incl(MPI_Group oldgroup,int n, int *ranks,MPI_Group *newgroup)
  ```

  ,
- создание новой группы **newgroup** из группы **oldgroup**, которая будет включать в себя n процессов, ранги которых не совпадают с рангами, перечисленными в массиве **ranks**:

  ```
  int MPI_Group_excl(MPI_Group oldgroup,int n, int *ranks,MPI_Group *newgroup).
  ```

Для получения новых групп над имеющимися группами процессов могут быть выполнены операции объединения, пересечения и разности:

- создание новой группы **newgroup** как объединения групп **group1** и **group2**:

  ```
  int MPI_Group_union(MPI_Group group1, MPI_Group group2, MPI_Group *newgroup)
  ```

  ;
- создание новой группы **newgroup** как пересечения групп **group1** и **group2**:

  ```
  int MPI_Group_intersection ( MPI_Group group1, MPI_Group group2,
  MPI_Group *newgroup ),
  ```

- создание новой группы **newgroup** как разности групп **group1** и **group2**:

  ```
  int MPI_Group_difference ( MPI_Group group1, MPI_Group group2,
  MPI_Group *newgroup ).
  ```

При конструировании групп может оказаться полезной специальная пустая группа MPI_COMM_EMPTY.

Ряд функций MPI обеспечивает получение информации о группе процессов:

- получение количества процессов в группе:

  ```
  int MPI_Group_size ( MPI_Group group, int *size ),
  ```

- получение ранга текущего процесса в группе:

  `int MPI_Group_rank ( MPI_Group group, int *rank ).`

После завершения использования группа должна быть удалена:

```
int MPI_Group_free ( MPI_Group *group )
```

(выполнение данной операции не затрагивает коммуникаторы, в которых используется удаляемая группа).

5.6.2. Управление коммуникаторами

Отметим прежде всего, что в данном пункте рассматривается управление **интракоммуникаторами**, используемыми для операций передачи данных внутри одной группы процессов. Как отмечалось ранее, применение **интеркоммуникаторов** для обменов между группами процессов выходит за пределы данного учебного материала.

Для создания новых коммуникаторов применимы два основных способа их получения:

- дублирование уже существующего коммуникатора:

  `int MPI_Comm_dup ( MPI_Comm oldcom, MPI_comm *newcomm ),`

- создание нового коммуникатора из подмножества процессов существующего коммуникатора:

  `int MPI_comm_create (MPI_Comm oldcom, MPI_Group group, MPI_Comm *newcomm).`

Дублирование коммуникатора может использоваться, например, для устранения возможности пересечения по тегам сообщений в разных частях параллельной программы (в т.ч. и при использовании функций разных программных библиотек).

Следует отметить также, что операция создания коммуникаторов является коллективной и, тем самым, должна выполняться всеми процессами исходного коммуникатора.

Для пояснения рассмотренных функций можно привести пример создания коммуникатора, в котором содержатся все процессы, кроме процесса, имеющего ранг 0 в коммуникаторе MPI_COMM_WORLD (такой коммуникатор может быть полезен для поддержки схемы организации параллельных вычислений "менеджер - исполнители" – см. раздел 6):

```
MPI_Group WorldGroup, WorkerGroup;
MPI_Comm Workers;
int ranks[1];
ranks[0] = 0;
// получение группы процессов в MPI_COMM_WORLD
MPI_Comm_group(MPI_COMM_WORLD, &WorldGroup);
// создание группы без процесса с рангом 0
MPI_Group_excl(WorldGroup, 1, ranks, &WorkerGroup);
// Создание коммуникатора по группе
MPI_Comm_create(MPI_COMM_WORLD,WorkerGroup,&Workers);
...
MPI_Group_free(&WorkerGroup);
MPI_Comm_free(&Workers);
```

Быстрый и полезный способ одновременного создания нескольких коммуникаторов обеспечивает функция:

```
int MPI_Comm_split ( MPI_Comm oldcomm, int split, int key,
MPI_Comm *newcomm ),
```

где oldcomm – исходный коммуникатор, split – номер коммуникатора, которому должен принадлежать процесс, key – порядок ранга процесса в создаваемом коммуникаторе, newcomm – создаваемый коммуникатор.

Создание коммуникаторов относится к коллективным операциям и, тем самым, вызов функции **MPI_Comm_split** должен быть выполнен в каждом процессе коммуникатора **oldcomm**. В результате выполнения функции процессы разделяются на непересекающиеся группы с одинаковыми значениями параметра **split**. На основе сформированных групп создается набор коммуникаторов. При создании коммуникаторов для рангов процессов выбирается такой порядок нумерации, чтобы он соответствовал порядку значений параметров **key** (процесс с большим значением параметра **key** должен иметь больший ранг).

В качестве примера можно рассмотреть задачу представления набора процессов в виде двумерной решетки. Пусть **p=q*q** есть общее количество процессов, следующий далее фрагмент программы обеспечивает получение коммуникаторов для каждой строки создаваемой топологии:

```
MPI_Comm comm;
int rank, row;
MPI_Comm_rank(MPI_COMM_WORLD,&rank);
row = rank/q;
MPI_Comm_split(MPI_COMM_WORLD,row,rank,&comm);
```

При выполнении данного примера, например, при **p=9**, процессы с рангами (0,1,2) образуют первый коммуникатор, процессы с рангами (3,4,5) – второй и т.д.

После завершения использования коммуникатор должен быть удален:

```
int MPI_Comm_free ( MPI_Comm *comm ).
```

5.7. Виртуальные топологии

Под **топологией** вычислительной системы обычно понимается структура узлов сети и линий связи между этими узлами. Топология может быть представлена в виде графа, в котором вершины есть процессоры (процессы) системы, а дуги соответствуют имеющимся линиям (каналам) связи.

Как уже отмечалось ранее, парные операции передачи данных могут быть выполнены между любыми процессами одного и того же коммуникатора, а в коллективной операции принимают участи все процессы коммуникатора. В этом плане, логическая топология линий связи между процессами в параллельной программе имеет структуру **полного графа** (независимо от наличия реальных физических каналов связи между процессорами).

Понятно, что физическая топология системы является аппаратно реализуемой и изменению не подлежит (хотя существуют и программируемые средства построения сетей). Но, оставляя неизменной физическую основу, мы можем организовать логическое представление любой необходимой **виртуальной топологии**. Для этого достаточно, например, сформировать тот или иной механизм дополнительной адресации процессов.

Использование виртуальных процессов может оказаться полезным в силу ряда разных причин. Виртуальная топология, например, может больше соответствовать имеющейся структуре линий передачи данных. Использование виртуальных топологий может заметно упростить в ряде случаев представление и реализацию параллельных алгоритмов.

В MPI поддерживаются два вида топологий - **прямоугольная решетка** произвольной размерности (**декартова топология**) и **топология графа** любого произвольного вида. Следует отметить, что имеющиеся в MPI функции обеспечивают лишь получение новых логических систем адресации процессов, соответствующих формируемым виртуальным топологиям. Выполнение же всех коммуникационных операций должно осуществляться, как и ранее, при помощи обычных функций передачи данных с использованием исходных рангов процессов.

5.7.1. Декартовы топологии (решетки)

Декартовы топологии, в которых множество процессов представляется в виде прямоугольной **решетки** (см. п. 1.4.1 и рис. 5.7), а для указания процессов используется декартова система координат, широко применяются во многих задачах для описания структуры имеющихся информационных зависимостей. В числе примеров таких задач – матричные алгоритмы (см. разделы 7 и 8) и сеточные методы решения дифференциальных уравнений в частных производных (см. раздел 12).

Для создания декартовой топологии (решетки) в MPI предназначена функция:

```
int MPI_Cart_create(MPI_Comm oldcomm, int ndims, int *dims, int *periods,
int reorder, MPI_Comm *cartcomm),
```

где: oldcomm - исходный коммуникатор, ndims - размерность декартовой решетки, dims - массив длины ndims, задает количество процессов в каждом измерении решетки, periods - массив длины ndims, определяет, является ли решетка периодической вдоль каждого измерения, reorder - параметр допустимости изменения нумерации процессов, cartcomm – создаваемый коммуникатор с декартовой топологией процессов.

Операция создания топологии является коллективной и, тем самым, должна выполняться всеми процессами исходного коммуникатора.

Для пояснения назначения параметров функции **MPI_Cart_create** рассмотрим пример создания двухмерной решетки **4x4**, в которой строки и столбцы имеют кольцевую структуру (за последним процессом следует первый процесс):

```
// создание двухмерной решетки 4x4
MPI_Comm GridComm;
int dims[2], periods[2], reorder = 1;
dims[0] = dims[1] = 4;
periods[0] = periods[1] = 1;
MPI_Cart_create(MPI_COMM_WORLD, 2, dims, periods, reoreder, &GridComm);
```

Следует отметить, что в силу кольцевой структуры измерений сформированная в рамках примера топология является **тором**.

Для определения декартовых координат процесса по его рангу можно воспользоваться функцией:

```
int MPI_Card_coords(MPI_Comm comm,int rank,int ndims,int *coords),
```

где: comm – коммуникатор с топологией решетки, rank - ранг процесса, для которого определяются декартовы координаты, ndims - размерность решетки, coords - возвращаемые функцией декартовы координаты процесса.

Обратное действие – определение ранга процесса по его декартовым координатам – обеспечивается при помощи функции:

```
int MPI_Cart_rank(MPI_Comm comm, int *coords, int *rank),
```

где comm – коммуникатор с топологией решетки, coords - декартовы координаты процесса, rank - возвращаемый функцией ранг процесса.

Полезная во многих приложениях процедура разбиения решетки на подрешетки меньшей размерности обеспечивается при помощи функции:

```
int MPI_Card_sub(MPI_Comm comm, int *subdims, MPI_Comm *newcomm),
```

где: comm - исходный коммуникатор с топологией решетки, subdims – массив для указания, какие измерения должны остаться в создаваемой подрешетке, newcomm - создаваемый коммуникатор с подрешеткой.

Операция создания подрешеток также является коллективной и, тем самым, должна выполняться всеми процессами исходного коммуникатора. В ходе своего выполнения функция **MPI_Cart_sub** определяет коммуникаторы для каждого сочетания координат фиксированных измерений исходной решетки.

Для пояснения функции **MPI_Cart_sub** дополним ранее рассмотренный пример создания двухмерной решетки и определим коммуникаторы с декартовой топологией для каждой строки и столбца решетки в отдельности:

```
// создание коммуникаторов для каждой строки и столбца решетки
MPI_Comm RowComm, ColComm;
int subdims[2];
// создание коммуникаторов для строк
subdims[0] = 0; // фиксации измерения
subdims[1] = 1; // наличие данного измерения в подрешетке
MPI_Cart_sub(GridComm, subdims, &RowComm);
// создание коммуникаторов для столбцов
subdims[0] = 1;
subdims[1] = 0;
MPI_Cart_sub(GridComm, subdims, &ColComm);
```

В приведенном примере для решетки размером 4x4 создаются 8 коммуникаторов, по одному для каждой строки и столбца решетки. Для каждого процесса определяемые коммуникаторы **RowComm** и **ColComm** соответствуют строке и столбцу процессов, к которым данный процесс принадлежит.

Дополнительная функция **MPI_Cart_shift** обеспечивает поддержку процедуры последовательной передачи данных по одному из измерений решетки (**операция сдвига данных** - см. раздел 3). В зависимости от периодичности измерения решетки, по которому выполняется сдвиг, различаются два типа данной операции:

- **Циклический сдвиг** на k элементов вдоль измерения решетки – в этой операции данные от процесса **i** пересылаются процессу **(i+k) mod dim**, где **dim** есть размер измерения, вдоль которого производится сдвиг,
- **Линейный сдвиг** на k позиций вдоль измерения решетки – в этом варианте операции данные от процессора **i** пересылаются процессору **i+k** (если таковой существует).

Функция **MPI_Cart_shift** обеспечивает получение рангов процессов, с которыми текущий процесс (процесс, вызвавший функцию **MPI_Cart_shift**) должен выполнить обмен данными:

```
int MPI_Card_shift(MPI_Comm comm, int dir, int disp,
int *source, int *dst),
```

где: comm – коммуникатор с топологией решетки, dir - номер измерения, по которому выполняется сдвиг, disp - величина сдвига (<0 – сдвиг к началу измерения), source – ранг процесса, от которого должны быть получены данные, dst - ранг процесса которому должны быть отправлены данные.

Следует отметить, что функция **MPI_Cart_shift** только определяет ранги процессов, между которыми должен быть выполнен обмен данными в ходе операции сдвига. Непосредственная передачами данных, может быть выполнена, например, при помощи функции MPI_Sendrecv.

5.7.2. Топологии графа

Сведения по функциям MPI для работы с виртуальными топологиями типа граф будут рассмотрены более кратко – дополнительная информация может быть получена, например, Немнюгин и Стесик (2002), Group, et al. (1994), Pacheco (1996).

Для создания коммуникатора с топологией типа граф в MPI предназначена функция

```
int MPI_Graph_create(MPI_Comm oldcomm, int nnodes, int *index, int *edges,
int reorder, MPI_Comm *graphcomm),
```

где: oldcomm - исходный коммуникатор, nnodes - количество вершин графа, index - количество исходящих дуг для каждой вершины, edges - последовательный список дуг графа, reorder - параметр допустимости изменения нумерации процессов, cartcomm – создаваемый коммуникатор с топологией типа граф.

Операция создания топологии является коллективной и, тем самым, должна выполняться всеми процессами исходного коммуникатора.

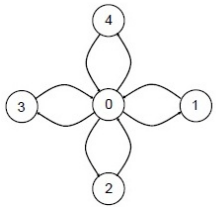

Рис. 5.9. Пример графа для топологии типа звезда

Для примера создадим топологию графа со структурой, представленной на рис. 5.9. В этом случае количество процессов равно 5, порядки вершин (количества исходящих дуг) принимают значения (4,1,1,1,1), а матрица инцидентности (номера вершин, для которых дуги являются входящими) имеет вид:

Процессы	Линии связи
0	1, 2, 3, 4
1	0
2	0
3	0
4	0

Для создания топологии с графом данного вида необходимо выполнить следующий программный код:

```
// создание топологии типа звезда
int index[] = { 4,1,1,1,1 };
int edges[] = { 1,2,3,4,0,0,0,0 };
MPI_Comm StarComm;
MPI_Graph_create(MPI_COMM_WORLD, 5, index, edges, 1, &StarComm);
```

Приведем еще две полезные функции для работы с топологиями графа. Количество соседних процессов, в которых от проверяемого процесса есть выходящие дуги, может быть получено при помощи функции:

```
int MPI_Graph_neighbors_count(MPI_Comm comm,int rank, int *nneighbors).
```

Получение рангов соседних вершин обеспечивается функцией:

```
int MPI_Graph_neighbors(MPI_Comm comm,int rank,int mneighbors, int *neighbors),
```

где **mneighbors** есть размер массива **neighbors**.