Использование средств MPI для реализации широковещательных сообщений и групповой пересылки при обмене данными.

Любую задачу в MPI можно решить, используя только связь типа «точка-точка», например, если необходимо передать значение вектора **x** всем процессам параллельной программы, то можно реализовать следующий код:

```
MPI_Comm_size(MPI_COMM_WORLD, &ProcNum);
for(int i = 1; i < ProcNum+1; i++)
   MPI_Send(&x, n, MPI_DOUBLE, i, 0, MPI_COMM_WORLD);
```

где **x** – массив вещественных чисел, **n**– размер массива. Однако такое решение неэффективно, поскольку повторение операций передачи приведет к суммированию затрат на подготовку передаваемых сообщений. Кроме того данная операция может быть выполнена за (ProcNum-1) итераций передачи данных. Более целесообразно использовать специальную MPI-функцию широковещательной рассылки, которая будет описана далее.

рПод коллективными операциями в MPI понимаются операции над данными, в которых принимают участие все процессы используемого коммуникатора. Из этого следует, что одним из ключевых аргументов в вызове коллективной функции является коммуникатор, который определяет группу (или группы) участвующих процессов, между которыми выполняется широковещательная передача данных. Несколько коллективных функций, таких как широковещательная рассылка (**broadcast**), сбор данных (**gather**) имеют единственный инициирующий или принимающий процесс, этот процесс называется корнем (**root**). Некоторые аргументы в коллективных функциях определены как аргументы, которые используются только корневым процессом, всеми другими участниками обмена эти аргументы игнорируются.

Условия согласования типов для коллективных операций являются более строгими, чем соответствующие условия между отправителем и получателем в передаче типе «точка-точка». Для коллективных операций количество отправленных данных должно точно совпадать с количеством данных, определенным приемником. Типы данных для гетерогенных распределенных систем, соответствующих типам данных MPI, могут отличаться.

Завершение коллективной операции может происходить, как только участие процесса в данной операции окончено. Завершение в таком случае указывает на то, что вызвавшая программа может изменять буфер передачи, но при этом это не значит, что другие процессы в группе завершили выполнение данной функции

или даже начали ее. Поэтому коллективная операция имеет некоторый эффект синхронизации всех вызывающих процессов. Процессы полностью синхронизирует выполнение функции барьера (**MPI_barrier**). Коллективные операций могут использовать в качестве аргумента те же коммуникаторы, что и коммуникации типа «точка-точка». MPI гарантирует, что созданные сообщения от имени коллективной передачи не будут пересекаться с сообщениями, созданными для передачи типа «точка-точка».

Коммуникатор и виды обмена

Ключевой особенностью коллективных функций является использование группы или групп участвующих процессов. Функции при этом не имеют явного идентификатора группы в качестве аргумента, для этой цели используется коммуникатор. Существуют два типа коммуникаторов:

- коммуникаторы процессов внутри одной группы(**intra-communicators**),
- коммуникаторы процессов для нескольких групп (**inter-communicators**).

В упрощённом варианте интра-коммуникатор – это обычный коммуникатор для одной группы процессов, соединенных контекстом, в то время как интеркоммуникатор определяет две отдельные группы процессов соединенных контекстом (см. рис. 1.1). Интер-коммуникатор позволяет передавать данные между процессами из разных интра-коммуникаторов (групп).

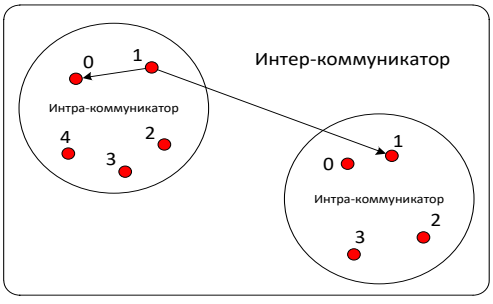

Рисунок 1.1 – Интра- и интер-коммуникаторы

Для выполнения коллективной операции (операции обмена) ее должны вызывать все процессы в группе, определенной интра-коммуникатором. В MPI используются коллективные операции интра-коммуникатора **All-To-All.** Каждый процесс в группе получает все сообщения от каждого процесса в этой же группе.

Функции, используемые при обмене: **MPI_Allgather**, **MPI_Allgatherv**; **MPI_Alltoall**, **MPI_Alltoallv**, **MPI_Alltoallw**; **MPI_Allreduce**, **MPI_Reduce_scatter**; **MPI_Barrier**.

All-To-One. Все процессы группы формируют данные, но лишь один принимает их. Функции, используемые при обмене: **MPI_Gather**, **MPI_Gatherv**, **MPI_Reduce**.

One-To-All. Один процесс группы формирует данные, все процессы этой же группы принимает его. Функции, используемые при обмене: **MPI_Bcast**, **MPI_Scatter**, **MPI_Scatterv**.

Коллективные коммуникации для интер-коммуникаторов легче всего описать для двух групп. К примеру, операция «все ко всем» (**MPI_Allgather**) может быть описана как сбор данных от всех членов одной группы и получение результата всеми членами другой группы (рис.1.2).

Для интра-коммуникаторов группа обменивающихся процессов одна и та же. Для интер-коммуникаторов они различны. Для операций «все ко всем», каждой такой операции соответствуют две фазы, так что она имеет симметрию, полнодуплексное поведение. Следующие коллективные операции также применяются для интер-коммуникаций: **MPI_Barrier**, **MPI_Bcast**, **MPI_Gather**, **MPI_Gatherv**, **MPI_Scatter**, **MPI_Scatterv**, **MPI_Allgather**, **MPI_Allgatherv**, **MPI_Alltoall**, **MPI_Alltoallv**, **MPI_Alltoallw**, **MPI_AllReduce**, **MPI_Reduce**, **MPI_Reduce_scatter.**

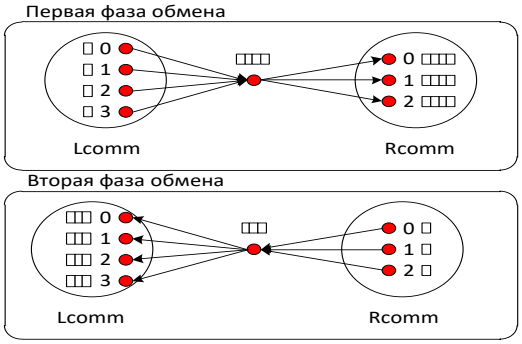

Рисунок 1.2 – Интер-коммуникатор **allgather**

Все процессы в обеих группах, определённых интер-коммуникатором, должны вызывать коллективные функции. Для коллективных операций интер-коммуникаторов если операция реализует обмен «все к одному» или «от одного ко всем», перемещение данных является однонаправленным. Направление передачи данных указывается в аргументе корня специальным значением. В этом

случае для группы, содержащей корневой процесс, все процессы должны вызывать функцию, использующую для корня специальный аргумент. Для этого корневой процесс использует значение **MPI_ROOT**; все другие процессы в той же группе, что и корневой процесс, используют **MPI_PROC_NULL**. Если операция находиться в категории «все ко всем», то перенос является двунаправленным.

Барьерная синхронизация

Для синхронизации выполнения всех процессов группы используется следующая функция:

Int MPI_Barrier (MPI_Comm comm);

Атрибутом этой функции является: in comm **– коммуникатор;**
Если аргумент **comm** является интра-коммуникатором, функция **MPI_Barrier** блокирует процесс, который его вызвал до того момента, пока все члены группы не вызовут эту же операцию. В любом процессе функция может завершиться только после того, как все члены группы достигли выполнения этой функции. Если аргумент **comm –** интер-коммуникатор, функция **MPI_Barrier** включает в себя две группы и завершается в процессах первой группы только после того, как все члены другой не начали выполнять данную функцию (и наоборот). При этом процесс может завершить выполнение данной функции до того, как все процессы в его собственной группе не начали ее выполнять.

Основные функции, реализующие широковещание

Чтобы передать значение от одного процесса всем процессам его группы (совершить **широковещательную рассылку**) используется функция, синтаксис которой имеет вид:

Int MPI_Bcast(void* buffer, int count, MPI_Datatype datatype, int root, MPI_Comm comm)

Атрибутами этой функции являются:
in out **buffer** - указатель на буфер; in **count** - количество элементов в буфере; in **datatype** - тип данных буфера; in **Root** - ранг корня широковещательной рассылки; in **Comm** – коммуникатор;

Если аргумент **comm** является интра-коммуникатор, функция **MPI_Bcast** распространяет сообщение от процесса с рангом **root** ко всем процессам группы, включая и себя. Это реализуется всеми членами группы, используя одни и те же аргументы **comm** и **root**. После завершения содержимое корневого буфера является скопированным во всех других процессах.

Рисунок 1.3 – Реализация функции широковещательной рассылки

Если аргумент **comm** является предварительно созданными интер-коммуникатором, тогда вызов включает в себя все процессы в интер-коммуникаторе, но при этом процесс-корень находиться в одной группе (группе А), а всем процессам другой группы (группы В) должен быть указан в качестве аргумента идентификатор корня (**root**) из группы, где находиться источник рассылки (группа А). Корневой процесс передает значение **MPI_ROOT** в аргумент **root**. Все другие процессы в группе А передают значение **MPI_PROC_NULL** в аргумент **root**. Данные распространяются от корня ко всем процессам в группе В. Для того чтобы собрать (**редукция**) значения со всех процессов группы в одном процессе, используется функция:

Int MPI_Gather(void* sendbuf, int sendcount, MPI_Datatypesendtype, void* recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)

Атрибутами этой функции являются:
in **sendbuf** - указатель на буфер отправки; in **sendcount** - количество элементов буфера отправки; in **sendtype** - тип данных буфера отправки; out **recvbuf** - указатель на буфер приема (используется только корневым процессом); in **recvcount** - количество элементов одиночного приема (неотрицательное число, используется только корневым процессом); in **recvtype** - тип данных буфера приема (используется только корневым процессом); in **root** - ранг принимающего процесса; in **comm** - коммуникатор;

Если аргумент **comm** интра-коммуникатор, тогда каждый процесс (включая корневой процесс) отправляют содержимое собственного буфера отправки

корневому процессу. Корневой процесс принимает сообщения и сохраняет их в ранжированном порядке. Вызов данной функции является идентичным тому, что каждый из **n** процессов в группе (включая коневой процесс) выполнили вызов **MPI_Send(sendbuf, sendcount, sendtype, root, …)**, а корень выполнил *n* вызовов **MPI_Recv().**

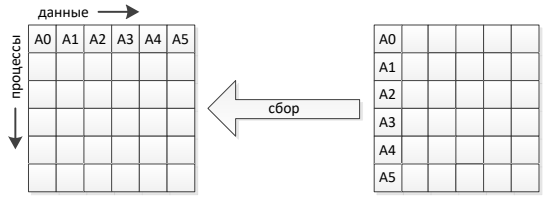

Рисунок 1.4 – Иллюстрация функции сбора

Если аргумент **comm** интер-коммуникатор, тогда вызов включает в себя все процессы интер-коммуникатора, но с одной группой (группа А), определяющей корневой процесс. Все процессы в другой группе (группа В) передают то же значение в аргумент **root**, который является корнем в группе А. В корневом процессе в качестве аргумента **root** указывается значение **MPI_ROOT**. Все другие процессы в группе А передают значение **MPI_PROC_NULL** в аргумент **root**. Данные собираются ото всех процессов в группе В к корню. Для сбора данных используется следующая функция:

Int MPI_Gatherv(void* sendbuf, intsendcount, MPI_Datatype sendtype, void* recvbuf, int *recvcounts, int *displs, MPI_Datatype recvtype, int root, MPI_Comm comm);

Атрибутами этой функции являются:
in **sendbuf** - указатель на буфер отправки; in **sendcount** - количество элементов буфера отправки; in **sendtype** - тип данных буфера отправки; out **recvbuf** - указатель на буфер приема; in **recvcounts** - массив неотрицательных чисел (длина равна размеру группы), содержащий количество элементов принимаемых от каждого процесса (используется только корневым процессом); in **displs** - целочисленный массив (длина равна размеру группы, элемент **i** указывает смещение относительно начала массива **recvbuf**, в котором необходимо заменить входные данные от процесса **i** (используется только корневым процессом)); in **recvtype** - Тип данных буфера приема (используется только корневым процессом); in **root** - Ранг принимающего процесса;

Так как аргумент **recvcounts** является массивом, функция **MPI_Gatherv** расширяет функциональность операции **MPI_Gather**, допуская переменное количество данных в каждом процессе. Дополнительную гибкость дает аргумент **displs** для данных, размещаемых в корне.

Пример использования функции **MPI_Gather().**

Необходимо произвести сбор 100 целочисленных значений от каждого процесса в группе к корню (рис. 1.5):

```
MPI_Comm comm;
int gsize, sendarray[100];
int root, *rbuf;
// ...
MPI_Comm_size(comm, &gsize);
rbuf = (int*)malloc(gsize*100*sizeof(int));
MPI_Gather(sendarray, 100, MPI_INT, rbuf, 100, MPI_INT, root, comm);
```

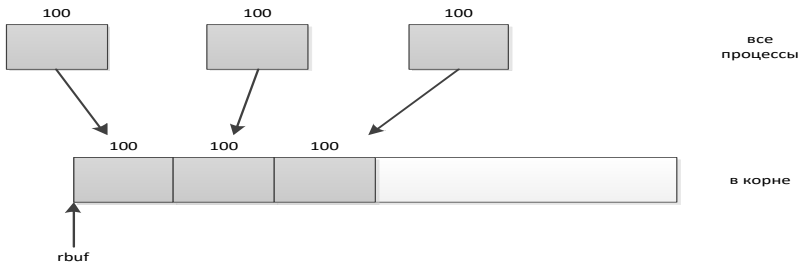

Рисунок 1.5 – Схема реализации сбора значений

Обратная функция для операции **MPI_Gather** следующая:

Int MPI_Scatter(void* sendbuf, intsendcount, MPI_Datatype sendtype, void* recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm);

Атрибутами этой функции являются:
in **sendbuf** - указатель на буфер отправки; in **sendcount** - количество элементов буфера отправки; in **sendtype** - тип данных буфера отправки; out **recvbuf** - указатель на буфер приема (используется только корневым процессом); in **recvcount** - количество элементов одиночного приема (используется только коневым процессом); in **recvtype** - тип данных буфера приема (используется только корневым процессом); in **root** - ранг принимающего процесса; in comm - Коммуникатор;

Если аргумент **comm** является интра-коммуникатор, результат можно проинтерпретировать так, как будто корень выполнил **n** операций отправки:

MPI_Send(sendbuf +i * sendcount * extent(sendtype), sendcount, sendtype, i, …);
и каждый процесс выполнил прием:

MPI_Recv(recvbuf, recvcount, recvtype, i, …);
Альтернативное описание является следующим: корень отсылает сообщение с помощью

MPI_Send(sendbuf, sendcount * n, sendtype, …);

Это сообщение расщепляется на **n** равных сегментов, **i**-ый сегмент отправляется **i**-ому процессу в группе, и каждый процесс принимает сообщение так, как описано выше. Буфер отправки игнорируется для всех некорневых процессов.

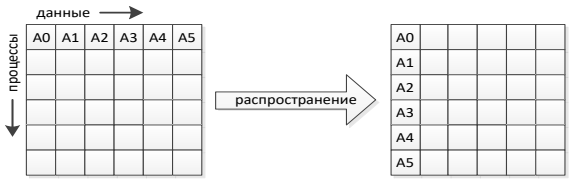

Рисунок 1.6 – Иллюстрация функции распространения MPI_Scatter ()

Если аргумент **comm** является интер-коммуникатором, тогда вызов включает в себя все процессы интер-коммуникатора, но только в одной группе (группа A) определяется корневой процесс. Все процессы в другой группе (группа B) передают одно и то же значение в аргумент **root**, которое является рангом корневого процесса в группе A. Корень передает значение **MPI_ROOT** в аргумент **root**. Данные распространяются от корня ко всем процессам в группе B. Чтобы совершить действия, обратные операции **MPI_Gatherv**, используется функция, синтаксис которой следующий:

int MPI_Scatterv(void* sendbuf, int *sendcounts, int *displs, MPI_Datatype sendtype, void* recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm);

Атрибутами этой функции являются:
in **sendbuf** - указатель на буфера отправки; in **sendcount** - количество элементов буфера отправки; in **displs** - целочисленный массив (длина–размер группы, элемент **i** указывает смещение относительно **sendbuf**, у которого необходимо взять данные, передаваемые процессу **I);** in **sendtype** - тип данных буфера отправки; out **recvbuf** - указатель на буфер приема; in **recvcount** - количество элементов буфера приема; in **recvtype** - тип данных буфера приема; in **root** - ранг принимающего процесса; in **comm** – коммуникатор;

Так как **sendcounts** является массивом, функция **MPI_Scatterv** расширяет функциональность **MPI_Scatter**, допуская переменное количество данных для отправки каждому процессу. Также дает дополнительную гибкость аргумент **displs**, указываемый для данных, которые берутся в корне.

Для того чтобы передать данные всем процессам, а не одному корню (как в функции **MPI_Gather**), используется следующая функция:

Int MPI_Allgather(void* sendbuf, int sendcount, MPI_Datatype sendtype, void* recvbuf, int recvcount, MPI_Datatype recvtype, MPI_Comm comm);

Атрибутами этой функции являются:

in **sendbuf** - указатель на буфер отправки; in **sendcount** - количество элементов буфера отправки; in **sendtype** - тип данных буфера отправки; out **recvbuf** - указатель на буфер приема; in **recvcount** - количество элементов буфера приема; in **recvtype** - тип данных буфера приема; in **comm** – коммуникатор.

Блок данных, отправленных от **j**-ого процесса, принимается каждым процессом и размещается в **j**-ом блоке буфера **recvbuf**. Если аргумент **comm** является интра-коммуникатором, результат вызова функции **MPI_Allgather** можно считать таким, что все процессы выполнили **n** вызовов: **MPI_Gather(sendbuf, sendcount, sendtype, recvbuf, recvcount, recvtype, root, comm);** где аргумент root принимает значения от 0 до n-1. Правила правильного использования функции **MPI_Allgather** аналогичны соответствующим правилам использования функции **MPI_Gather**.

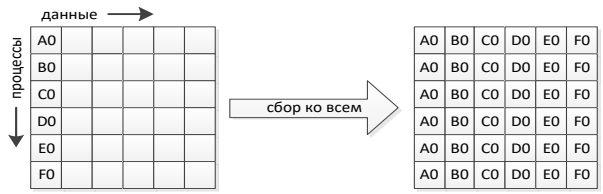

Рисунок 1.7 – Иллюстрация функции сбора ко всем

Если необходимо передать различное количество данных всем процессам, то используется функция, синтаксис которой следующий:

Int MPI_Allgatherv(void* sendbuf, int sendcount, MPI_Datatype sendtype, void* recvbuf, int recvcounts, int *displs, MPI_Datatype recvtype, MPI_Comm comm)

Атрибутами этой функции являются:
in **sendbuf** - указатель на буфер отправки; in **sendcount** - количество элементов буфера отправки; in **sendtype** - тип данных буфера отправки; out recvbuf - указатель на буфер приема; in **recvcounts** - количество элементов буфера приема; in **displs** -цЦелочисленный массив (длина равна размеру группы). Элемент **I** указывает смещение относительно **recvbuf**, где необходимо поместить входные данные от процесса **I;** in recvtype - Тип данных буфера приема; in comm – Коммуникатор;

Блок данных, отправленных от **j**-ого процесса, принимается всеми процессами и размещается в **j**-ом блоке буфера **recvbuf**. Для того чтобы каждый процессом отправил различные данные каждому процессу-приемнику, используется следующая функция:

Int MPI_Alltoall(void* sendbuf, int sendcount, MPI_Datatype sendtype, void* recvbuf, int recvcount, MPI_Datatype recvtype, MPI_Comm comm);
Атрибутами этой функции являются:

in **sendbuf** - указатель на буфер отправки; in **sendcount** - количество элементов буфера отправки; in **sendtype** - тип данных буфераотправки; out **recvbuf** - указатель на буфер приема; in **recvcount**- количество элементов буфера приема; in **recvtype** - тип данных буфера приема; in **comm** – коммуникатор;

Функция **MPI_Alltoall** является расширением функции **MPI_Allgather**, где **j**-ый блок, отправленный от **i**-ого процесса, принимается **j**-ым процессом и размещается в **i**-ом блоке **recvbuf**. Все аргументы на всех процесса являются значимыми. Аргумент **comm** должен иметь идентичное значение на всех процессах.

Рисунок 1.8 – Иллюстрация функции полного обмена

Если аргумент **comm** является интер-коммуникатором, тогда результат таков, что каждый процесс группы A отправляет сообщение каждому процессу группы B и наоборот.

Для того чтобы каждый процессом отправил данные различного размера каждому процессу-приемнику, используется следующая функция:

Int MPI_Alltoallv(void*sendbuf, int*sendcounts,int *sdispls,MPI_Datatypesendtype, void*recvbuf, int *recvcounts,int*rdispls,MPI_Datatype recvtype,MPI_Comm comm);

Атрибутами этой функции являются:
in **sendbuf** - указатель на буфер отправки; in **sendcounts** - массив неотрицательных целочисленных значений (длина равна размеру группы) указывающий количество элементов, которые необходимо отправить каждому процессу; in **sdispls** - целочисленный массив (длина – размер группы, элемент **j** указывает смещение относительно аргумента **sendbuf**, у которого необходимо взять данные передаваемые процессу **j);** in **sendtype** - тип данных буфера отправки; out **recvbuf** - указатель на буфер приема; in **recvcounts** - массив неотрицательных чисел (длина равна размеру группы), указывающий количество элементов, которое может быть принято от каждого процесса; in **rdispls** - целочисленный массив (длина равна размеру группы, элемент **i** указывает смещение относительно аргумента **recvbuf**, где необходимо поместить входные данные от процесса **I);** in **recvtype** - тип данных буфера приема; in **comm** – коммуникатор;

Основная концепция работы с несколькими процессами

Группа является упорядоченным набором идентификаторов процессных (далее процессы). Каждый процесс в группе ассоциируется с целочисленным рангом. Ранги являются последовательными и начинаются с нуля. Группа используется внутри коммуникатора для описания участников коммуникационной «среды» и для ранжирования этих участников (т.е. происходит раздача им уникального имени внутри «среды» коммуникации).

Существует специальная предопределенная группа: **MPI_GROUP_EMPTY**, которая является группой без участников. Предопределённая константа **MPI_GROUP_NULL** является значением, используемым для недопустимого идентификатора группы.

MPI-коммуникационные операции ссылаются на коммуникаторы для определения границ и «коммуникационной среды», в которой операции типа точка-точка или коллективного типа являются возможными. Каждый коммуникатор содержит группу допустимых участников; эта группа всегда включает в себя локальный процесс. Источник и место назначения сообщения определяются рангом процесса внутри группы. Для коллективных коммуникаций интра-коммуникатор определяет ряд процессов, которые участвуют в коллективной операции (и их порядок, когда это необходимо). Поэтому коммуникатор ограничивает «пространственные» границы коммуникации, и обеспечивает адресацию процессов независящую от машины посредством ранга.

После инициализации локальный процесс может общаться со всеми процессами интра-коммуникатора **MPI_COMM_WORLD** (включая и себя), определённого один раз **MPI_INIT** или **MPI_INIT_THREAD** вызовами. Предопределенная константа **MPI_COMM_NULL** является значением, использующимся для недопустимого дескриптора коммуникатора.

В реализации статической модели процессов MPI, все процессы, которые участвуют в общении, являются доступными после инициализации. Для этого случая **MPI_COMM_WORLD** является коммуникатором всех процессов доступных для коммуникации. В реализации MPI, процессы начинают вычисления без доступа ко всем другим процессам. В таком случае **MPI_COMM_WORLD** является коммуникатором, объединяющим все процессы, с которыми присоединяющийся процесс может незамедлительно общаться. По этой причине **MPI_COMM_WORLD** может одновременно представлять отделенные группы в разных процессах.

Коммуникатор **MPI_COMM_WORLD** не может быть освобождён в течение функционирования процесса. Группа соответствующая этому коммуникатору

нигде не проявляется, так как является предопределенной константой, но она может быть доступна через использование функции **MPI_Comm_Group**.

Управление группами

Чтобы получить информацию о группе, используются следующие функции:

int MPI_Group_size(MPI_Group group, int *size).

Атрибутами этой функции являются:
in **group** – группа; out **size** - количество процессов в группе.

int MPI_Group_rank(MPI_Group group, int *rank).

Атрибутами этой функции являются:
in **group** – группа; out **rank** - ранг вызывающего процесса в группе, или значение **MPI_UNDEFINED**,если процесс не является членом группы.

Следующая функция используется для определения относительной нумерации одних и тех же процессов в двух разных группах:

int MPI_Group_translate_ranks (MPI_Group group1, int n, int *ranks1, MPI_Group group2, int *ranks2)

Атрибутами этой функции являются:
in **group1** - первая группа; in **n** - количество рангов в массивах **ranks1**и **ranks2;** in **ranks1** - массив нулевого или более допустимых рангов в **group1;** in **group2** - вторая группа; out **ranks2** - массив соответствующих рангов в **group2**, значение **MPI_UNDEFINED,** если нет соответствий;

К примеру, если известны ранги текущих процессов в группе **MPI_COMM_WORLD**, может понадобиться узнать их ранги в подмножестве этой группы. Значение**MPI_PROC_NULL**является допустимым рангом для ввода в **MPI_Group_translate_ranks**, который возвращает значение **MPI_PROC_NULL** как переданный ранг.

Конструкторы группы используются для создания подмножества и расширения существующих групп. Эти конструкторы создают новые группы из существующих групп. Существуют локальные операции и отдельные группы, которые могут быть определены на разных процессах; процесс может также определять группу, которая не включает себя. Устойчивое определение необходимо, когда группы используются как аргументы в функциях создания коммуникаторов. MPI не обеспечивает механизм для создания групп с нуля, их

можно создать только из других до этого определенных групп. Основная группа, с помощью которой строятся все остальные группы, является группа, ассоциированная с начальным коммуникатором **MPI_COMM_WORLD**, доступная через функцию **MPI_Comm_group**:

Int MPI_Comm_group(MPI_Commicomm, MPI_Group *group);

Атрибутами этой функции являются:
in **comm** – коммуникатор; out **group** –группа, соответствующая **comm;**

Следующие функции позволяют создавать новые группы:

Int MPI_Group_union(MPI_Groupgroup1, MPI_Groupgroup2,
MPI_Group *newgroup);

Атрибутами этой функции являются:
in group1 - первая группа; in group2 - вторая группа; out newgroup - группа объединения.

Int MPI_Group_intersection(MPI_Group group1, MPI_Group group2, MPI_Group *newgroup);

Атрибутами этой функции являются:
in **group1** - первая группа; in **group2** - вторая группа; out **newgroup** - группа пересечения.

int MPI_Group_difference(MPI_Group group1, MPI_Group group2,
MPI_Group *newgroup) **Атрибутами этой функции являются:**
in **group1** - первая группа; in **group2** - вторая группа; out **newgroup** - группа разницы;

Операции определены следующим образом:

1) в **MPI_Group_Union()** все элементы первой группы (**group1**), затем все элементы второй группы (**group2**), которых нет в первой группе.
2) в **MPI_Group_Intersection()** все элементы первой группы, которые также есть и во второй группе, упорядоченные как в первой группе.
3) в **MPI_Group_Difference()** все элементы первой группы, которых нет во второй группе, упорядоченные как в первой группе.

Для этих операций порядок процессов выходной группы определяется главным образом порядком в первой группе (если возможно) и , если необходимо, порядком во второй группе. Ни объединение, ни пересечение не являются коммутативными, но обе являются ассоциативными. Чтобы включить процессы в новую группу с определенными рангами используется следующая функция:

int MPI_Group_incl(MPI_Group group, int n, int *ranks,
MPI_Group *newgroup);

Атрибутами этой функции являются:

in **group** – группа; in **n** - количество элементов в массиве рангов (и размер **newgroup**); in **ranks** - ранги процессов в **group**, которые должны появиться в **newgroup;**out **newgroup** - новая группа производная от вышеупомянутой, в порядке определения **ranks;**

Функция **MPI_Group_incl()** создает группу **newgroup**, которая состоит из **n** процессов в **group** с рангами **ranks[0]**, …, **ranks[n-1]**; процесс с рангом **i** в **newgroup** является процессом с рангом **ranks[i]** в **group**. Каждый из **n** элементов ранга **ranks** должен быть допустимым рангом в **group**, и все элементы должны быть индивидуальными, иначе программа является ошибочной. Если **n = 0**, тогда группа **newgroup** является **MPI_GROUP_EMPTY**. Эти функции могут, к примеру, использоваться для переупорядочивания элементов группы, или для сравнения. Для того, чтобы исключить процессы из группы с определенными рангами используется следующая функция:

int MPI_Group_excl(MPI_Group group, int n, int *ranks,
MPI_Group *newgroup);

Атрибутами этой функции являются:

in **group** – группа; in **n** - количество элементов в массиве рангов (и размер **newgroup**); in **ranks** - массив целочисленных рангов в **group**, которые не должны появиться в **newgroup;** out **newgroup** - новая группа производная от вышеупомянутой, сохраняющая порядок определенный **group;**

Функция **MPI_Group_excl** создает группу процессов **newgroup** путем удаления из **group** этих процессов с рангами **ranks[0]**, …, **ranks[n-1]**. Упорядочивание процессов в **newgroup** является идентичным упорядочиванию в **group**. Каждый из **n** элементов рангов **ranks** должен быть допустимым в **group.** Если **n = 0**, тогда группа **newgroup** является идентичной **group**.

Управление коммуникаторами

Операции, которые получают доступ к коммуникаторам, являются локальными и их выполнение не требует взаимодействия процессов. Операции, которые создают коммуникаторы, являются коллективными и могут требовать взаимодействия между процессами. Для того чтобы сравнить два коммуникатора используется следующая функция:

int MPI_Comm_compare(MPI_Comm comm1, MPI_Comm comm2, int *result);

Атрибутами этой функции являются:

in **comm1** - первый коммуникатор; in **comm2** - второй коммуникатор; out **result** – результат.

Результат будет иметь значение **MPI_IDENT**, если аргументы **comm1** и **comm2** являются дескрипторами одного и того же объекта (идентичные группы и одинаковый контекст). Значение **MPI_CONGRUENT** будет результатом, если группы являются идентичными в компонентах и порядке рангов; эти коммуникаторы отличаются только контекстом. Значение **MPI_SIMILAR** является результатом, если члены группы обоих коммуникаторов одинаковы, но порядок рангов отличается. Результат будет равен значению **MPI_UNEQUAL** в любом другом случае. Следующие операции являются коллективными функциями, которые вызываются всеми процессами в группе или группах ассоциированных с аргументом **comm**.

MPI обеспечивает четыре функции конструирования коммуникатора, которые применяются к интра-коммуникаторам и интер-коммуникаторам. Функция конструирования **MPI_Intercomm_create** применяется только к интер-коммуникаторам. В интер-коммуникаторе группы называются левой и правой. Процесс в интер-коммуникаторе является членом либо левой, либо правой группы. С точки зрения этого группа процесса, членом которой он является, называется локальной; другая группа (относительно этого процесса) является удаленной (дистанционной). Метки левой и правой групп дают возможность указывать две группы в интер-коммуникаторе, которые не относятся к какому-либо конкретному процессу.

Для создания коммуникатора используется следующая функция:

Int MPI_Comm_create(MPI_Comm comm, MPI_Group group, MPI_Comm *newcomm) **;**

Атрибутами этой функции являются:
in **comm** – коммуникатор; in **group** - группа, которая является подмножеством группы; коммуникатора **comm;** out **newcomm** - новый коммуникатор**.**

Если аргумент **comm** является интра-коммуникатором, эта функция возвращает новый коммуникатор **newcomm** с коммуникационной группой, определённой аргументом **group**. Никакая кэшированная информации не распространяется от коммуникатора **comm** к новому коммуникатору **newcomm**. Каждый процесс должен выполнять вызов функции с аргументом **group**, который является подгруппой группы ассоциированной с аргументом **comm**. Процессы могут указывать различные значения для аргумента **group**.

Если аргумент **comm** является интер-коммуникатором, тогда выходной коммуникатор также является интер-коммуникатором, где локальная группа состоит только из тех процессов, которые содержатся в группе **group**. Аргумент **group** должен иметь те процессы в локальной группы входного интер-коммуникатора, который является частью коммуникатора **newcomm**. Все

процессы в одной и той же локальной группе коммуникатора **comm** должны указывать одинаковое значение для аргумента **group**. Если группа **group** не указывает хотя бы один процесс в локальной группе интер-коммуникатора, или если вызывающий процесс не является включенным в аргумент **group**, возвращается **MPI_COM_NULL**.

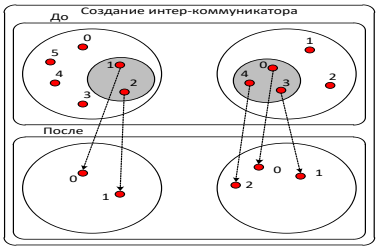

Рисунок 1.10 – Создание интер-коммуникатора

Для разделения коммуникатора используется следующая функция:

Int MPI_Comm_split(MPI_Commcomm, int color, int key, MPI_Comm *newcomm);

Атрибутами этой функции являются:
in **comm** – коммуникатор; in **color** - контроль распределения подмножества; in **key** - контроль распределения рангов; out **newcomm** - новый коммуникатор.

Функция разделяет группу, ассоциированную с аргументом **comm** на отдельные подгруппы, одна для каждого значения аргумента **color**. Каждая подгруппа содержит все процессы одинакового цвета. Внутри каждой группы процессы ранжируются в порядке определенном значением аргумента **key**, со связями, соответствующими их рангу в старой группе. Новый коммуникатор создается для каждой подгруппы и возвращается в аргументе **newcomm**. Процесс может обеспечивать значение аргумента color как **MPI_UNDEFINED**, в таком случае коммуникатор **newcomm** будет иметь значение **MPI_COMM_NULL**. Это коллективный вызов, но каждому процессу разрешается задавать различное значения **color** и **key**.

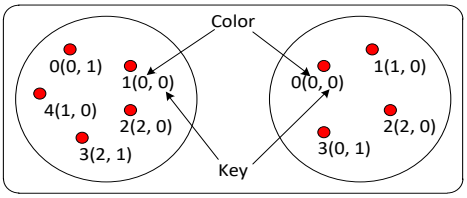

Рисунок 1.11 – Входной коммуникатор (**comm**)

С интра-коммуникатором **comm** вызов **MPI_Comm_create(comm, group, newcomm)** является эквивалентом вызова **MPI_Comm_split(comm, color, key, newcomm)**, где процессы, которое являются членами их аргумента **group**

обеспечивают **color**= числу групп (основывается на уникальной нумерации всех отдельных групп) и **key** = рангу в группе, все процессы которые не являются членами их аргумента **group** обеспечивают **color**= **MPI_UNDEFINED**.

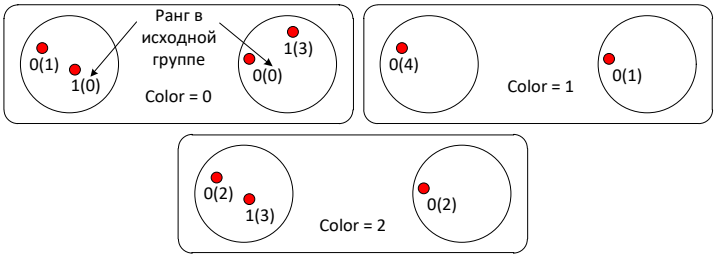

Рисунок 1.12 – Отделенные выходные коммуникаторы (**newcomm**)

Результат функции **MPI_Comm_split** на интер-коммуникаторе является таким, что процессы слева с одинаковым значением аргумента **color**, как у процессов справа, объединяются, чтобы создать новый интер-коммуникатор. Аргумент **key** описывает относительный ранг процессов на каждой стороне интер-коммуникатора. Для тех цветов, которые указываются только на одной стороне интер-коммуникатора, возвращается значение **MPI_COMM_NULL**. Для освобождения коммуникатора используется функция, синтаксис которой следующий:

Int MPI_Comm_free(MPI_Comm *comm)**;**

Атрибутом этой функции является: in comm **- коммуникатор, который необходимо освободить.**

Дескриптор устанавливается в значение **MPI_COMM_NULL**. Любые неоконченные операции, которые используют этот коммуникатор, будут завершены в нормальном режиме; объект освобожден фактически, только если не существует активных ссылок на него. Этот вызов применяется для интра- и интер-коммуникаторов. Функция **MPI_Intercomm_create** используется для того, чтобы связать два интра-коммуникатора в интер-коммуникатор. Функция **MPI_Intercomm_merge** создает интра-коммуникатор, соединяя локальную и удаленную группы интер-коммуникатора. Частичное совпадение локальной и удаленной групп, которые связаны в интер-коммуникаторе, запрещено. Если существует частичное совпадение, то тогда программа является ошибочной и вероятней всего ведет к взаимной блокировке.

Функция **MPI_Intercomm_create** может быть использована для создания интер-коммуникатора из двух существующих интра-коммуникаторов в следующей ситуации: хотя бы один выбранный член из каждой группы («лидер группы») имеет возможность взаимодействовать с выбранным членом из другой группы;

другими словами существует так называемый равноправный коммуникатор (“peer” communicator), которому принадлежат оба лидера, и каждый лидер знает ранг другого лидера в этом равноправном коммуникаторе. Кроме того члены каждой группы знают ранг их лидера.

Построение интер-коммуникатора из двух интра-коммуникаторов требует вызовов отдельных коллективных операций в локальной группе и в удаленной группе, также как передача типа «точка-точка» между процессом в локальной группе и процессом в удалённой группе.

Алгоритм создания интер-коммуникатора (для локальных групп с последующим обменом между ними): 1) формирование групп процессов для коммуникатора по умолчанию (**MPI_COMM_WORLD**); 2) исключение процессов из групп (формирование ограниченных групп процессов); 3) создание коммуникатора для обмена внутри группы (связывание интра-коммуникатора с каждой из групп); 4) создание интер-коммуникатора.

Синтаксис функции создания интер-коммуникатора следующий:

int MPI_Intercomm_create(MPI_Comm local_comm, int local_leader, MPI_Comm peer_comm, int remote_leader, int tag,
MPI_Comm *newintercomm);

Атрибутами этой функции являются:
in **local_comm** - локальный интра-коммуникатор; in **local_leader** - ранг лидера локальной группы в коммуникаторе **local_comm;**in **peer_comm** - Равноправный коммуникатор; используется только процессом **local_leader; i**n **remote_leader** - ранг лидера удаленной группы в коммуникаторе **peer_comm**; используется только процессом **local_leader;** In **tag** - метка «безопасности»; out **newintercomm** - новый интер-коммуникатор.

Вызов данной функции является коллективным через объединение локальной и удаленной групп. Процессы должны обеспечивать одинаковые аргументы **local_comm** и **local_leader** внутри каждой группы. Групповое значение не разрешено для аргументов **remote_leader**, **local_leader** и **tag**.

Этот вызов использует коммуникацию типа «точка-точка» с коммуникатором **peer_comm** и с меткой **tag**.

Для создания интра-коммуникатора используется следующая функция:

int MPI_Intercomm_merge(MPI_Comm intercomm, int high,
MPI_Comm *newintracomm);

Атрибутами этой функции являются:
in **intercom** - Интер-коммуникатор; in **high** – флаг; out **newintracomm** - новый интра-коммуникатор.

Эта функция создает интра-коммуникатор из объединения двух групп, которые ассоциированы с коммуникатором **intercomm**. Все процессы должны иметь одинаковое значение **high** внутри каждой группы. Если процессы в одной группе указывают значение **false** для аргумента **high**, и процессы в другой указывают значение **true** для аргумента **high**, тогда объединение упорядочивается так, что «нижняя» группа располагает до «верхней» группы. Если все процессы указывают одинаковое значение для аргумента **high**, тогда порядок в объединении является произвольным. Вызов данной функции является блокирующимся и коллективным внутри объединения двух групп.