Интерфейс передачи сообщений **MPI**

- Стандарт механизма передачи сообщений MPI (Message Passing Interface) был принят в 1994 г. Стандарт готовился с 1992 по 1994 годы. группой Message Passing Interface Forum. Основная цель, которую ставили перед собой разработчики MPI – обеспечение полной независимости программ, написанных с использованием библиотеки MPI, от архитектуры многопроцессорной системы.
- Отметим, что интерфейс громоздок и сложен для прикладного программиста. Интерфейс также сложен для реализации. Поэтому практически отсутствуют реализации MPI, в которых в полной мере обеспечивается совмещение обменов с вычислениями.
- В 1997 появился проект стандарта MPI-2, который выглядит еще более громоздким и трудным для полной реализации. Реализацией стандарта MPI-2 является библиотека MPI-2.

Общие сведения о MPI

- стандарт MPI не стандартизует реализацию. Поставщики многопроцессорных вычислительных систем, как правило, предлагают свои реализации стандарта MPI для своих машин. Однако правильная *MPI-программа* (программа, написанная с использованием MPI) должна одинаково выполняться на всех реализациях.
- Существуют стандартные "привязки" MPI к языкамСуществуют стандартные "привязки" MPI к языкам C/C++, Fortran 77/90, а также бесплатные и коммерческие реализации MPI почти для всех суперкомпьютерных платформ, а также для UNIX и Windows NT вычислительных кластеров. Переносимой версией библиотеки MPI является библиотека MPICH.
- MPI-программаMPI-программа представляет собой набор независимых процессовMPI-программа представляет собой набор независимых процессов, каждый из которых выполняет свою собственную программу, написанную на каком-либо языкеMPI-программа представляет собой набор независимых процессов, каждый из которых выполняет свою собственную программу, написанную на каком-либо языке высокого уровня. Процессы MPI-программы

- MPI-библиотека реализована в виде примерно 130 функций, в число которых входят следующие основные наборы функций:
- функции для работы с *группами процессов* и *коммуникаторами*;
- функции, реализующие коммуникационные операции типа точка-точка;
- функции, реализующие коллективные коммуникационные операции.

Набор функций библиотеки MPIНабор функций библиотеки MPI далеко выходит за рамки набора функций, минимально необходимого для поддержки механизма передачи сообщений. Почти любая MPI-программа может быть написана с использованием всего 6 MPI-функций, а достаточно полную и удобную среду программирования составляет набор из 24 функций.

- MPI не обеспечивает механизмов задания начального размещения процессовMPI не обеспечивает механизмов задания начального размещения процессов по процессорам. Предполагается, что для этого имеются другие средства. Эти средства должны позволять задать число процессоров и процессов, разместить процессы и данные на каждом процессоре.
- Стандарт MPI требует, чтобы MPI-программыСтандарт MPI требует, чтобы MPI-программы могли выполняться на гетерогенных многопроцессорных системахСтандарт MPI требует, чтобы MPI-программы могли выполняться на гетерогенных многопроцессорных системах, в которых различные процессорыСтандарт MPI требует, чтобы MPI-программы могли выполняться на

Основными понятиями стандарта MPI являются:

- процесс;
- группа процессов;
- коммуникатор.
- ПроцессПроцесс – исполнение программы на одном процессоре. Процесс может содержать последовательный код, параллельные ветви, операции ввода/вывода и пр.
- Группа процессовГруппа процессов – совокупность процессовГруппа процессов – совокупность процессов, каждый из которых имеет внутри группы уникальное имя. Процессы в группе взаимодействуют посредством коммуникатора группы. Процессы в группе имеют номер от 0 до (n-1), где n – количество процессов в группе.
- КоммуникаторКоммуникатор реализует обмены данными между процессами и их синхронизацию. Для прикладной программы коммуникатор выступает в качестве коммуникационной среды. Различают внутригрупповые коммуникаторы (intra) и межгрупповые коммуникаторы

Программа "Hello world!

```
#include <stdio.h>
#include <mpi.h>

void main(int argc , char *argv[])
{
    MPI_Init( &argc, &argv );
    printf("Hello world!");
    MPI_Finalize();
}
```

Модель параллельной программы в ***MPI***

- В модели программирования MPI параллельная программа при запуске порождает несколько процессов, взаимодействующих между собой с помощью сообщений. Совокупность всех процессов, составляющих параллельное приложение, или их части, описывается специальной структурой, которая называется **коммуникатором** (областью взаимодействия).
- Каждому процессу в области взаимодействия назначается уникальный числовой идентификатор - **ранг**, значение которого от 0 до **np** – 1 (np - число процессов).

- Структура программы, написанной по схеме хозяин/работник, приведена ниже.

```
if (ранг процесса = рангу мастер-процесса)
{
        код мастер-процесса
} else
{
        код подчиненного процесса (подчиненных процессов)
}
```

- Каждый экземпляр программы уже в процессе своего выполнения определяет, является ли он мастер-процессом. Затем, в зависимости от результата этой проверки, выполняется одна из ветвей условного оператора. Первая ветвь соответствует мастер-задаче, а вторая - подчиненной задаче. Способы взаимодействия между подзадачами определяются программистом. Перед использованием процедур передачи сообщений программа должна подключиться к системе обмена сообщениями.

Сообщения

Сообщение содержит пересылаемые данные и служебную информацию. Для того, чтобы передать сообщение, необходимо указать:

- ранг процесса-отправителя сообщения;
- адрес, по которому размещаются пересылаемые данные процесса-отправителя;
- тип пересылаемых данных;
- количество данных;
- ранг процесса, который должен получить сообщение;
- адрес, по которому должны быть размещены данные процессом-получателем.
- тег сообщения;
- идентификатор коммуникатора, описывающего область взаимодействия, внутри которой происходит обмен.

- Тег - это задаваемое пользователем целое число от 0 до 32767, которое играет роль идентификатора сообщения и позволяет различать сообщения, приходящие от одного процесса. Теги могут использоваться и для соблюдения определенного порядка приема сообщений.

  Прием сообщения начинается с **подготовки буфера достаточного размера**. В этот буфер записываются принимаемые данные. Операция отправки или приема сообщения считается завершенной, если программа может вновь использовать буферы сообщений.

Разновидности обменов сообщениями

- В MPI реализованы разные виды обменов. Прежде всего, это **двухточечные** (задействованы только два процесса) и **коллективные** (задействованы более двух процессов).
- Двухточечные обмены используются для организации локальных и неструктурированных коммуникаций.
- При выполнении глобальных операций используются коллективные обмены.
- **Асинхронные** коммуникации реализуются с помощью запросов о получении сообщений.

Имеется несколько разновидностей двухточечного обмена.

- **Блокирующие** прием/передача - приостанавливают выполнение процесса на время приема сообщения.
- **Неблокирующие** прием/передача - выполнение процесса продолжается в фоновом режиме, а программа в нужный момент может запросить подтверждение завершения приема сообщения.
- ***Синхронный обмен*** - сопровождается уведомлением об окончании приема сообщения.
- **Асинхронный обмен** - уведомлением не сопровождается.

MPI & C/C++

- В программах на языке С имена функций имеют вид
- Класс_действие_подмножество или Класс_действие. В С++ подпрограмма является методом для определенного класса, имя имеет в этом случае вид MPI::Класс:: действие_подмножество. Для некоторых действий введены стандартные наименования: Create - создание нового объекта, Get – получение информации об объекте, Set - установка параметров объекта, Delete – удаление информации, Is - запрос о том, имеет ли объект указанное свойство.
- Имена констант MPI записываются в верхнем регистре. Их описания находятся в заголовочном файле mpi.h.
- Входные параметры функций передаются по значению, а выходные (и INOUT) — по ссылке.
- Соответствие типов MPI стандартным типам языка С приведено в табл.

Тип данных MPI	Тип данных C
MPI_CHAR	Signed char
MPI_SHORT	Signed short int
MPI_INT	Signed int
MPI_LONG	Signed long int
MPI_UNSIGNED_CHAR	unsigned char
MPI_UNSIGNED_SHORT	unsigned short int
MPI_UNSIGNED	unsigned int
MPI_UNSIGNED_LONG	unsigned long int
MPI_FLOAT	Float
MPI_DOUBLE	Double
MPI_LONG_DOUBLE	long double
MPI_BYTE	Нет соответствия
MPI_PACKED	Нет соответствия

Коды завершения

- В MPI приняты стандартные соглашения о кодах завершения вызовов подпрограмм. Так, например, возвращаются значения MPI_SUCCESS - при успешном завершении вызова и MPI_ERR_OTHER - обычно при попытке повторного вызова MPI_Init.

Вместо числовых кодов в программах обычно используют специальные именованные константы:

- MPI_ERR_BUFFER - неправильный указатель на буфер;
- MPI_ERR_COMM - неправильный коммуникатор;
- MPI_ERR_RANK - неправильный ранг;
- MPI_ERR_OP - неправильная операция;
- MPI_ERR_ARG - неправильный аргумент;
- MPI_ERR_UNKNOWN - неизвестная ошибка;
- MPI_ERR_TRUNCATE - сообщение обрезано при приеме;
- MPI_ERR_INTERN - внутренняя ошибка. Обычно возникает, если системе не хватает памяти.

- Коммуникатор представляет собой структуру, содержащую либо все процессы, исполняющиеся в рамках данного приложения, либо их подмножество.
- Процессы, принадлежащие одному и тому же коммуникатору, наделяются общим контекстом обмена. Операции обмена возможны только между процессами, связанными с общим контекстом, то есть, принадлежащие одному и тому же коммуникатору. Каждому коммуникатору присваивается идентификатор. В MPI есть несколько стандартных коммуникаторов:
- MPI_COMM_WORLD – включает все процессы параллельной программы;
- MPI_COMM_SELF – включает только данный процесс;
- MPI_COMM_NULL – пустой коммуникатор, не содержит ни одного процесса.
- В MPI имеются процедуры, позволяющие создавать новые коммуникаторы, содержащие подмножества процессов.

Коммуникатор и ранги

Функции MPI

int MPI_Init(int *argc, char **argv)

Подключение к MPI. Данный вызов предшествует всем прочим вызовам подпрограмм MPI.

int MPI_Finalize()

Завершение работы с MPI. После вызова данной подпрограммы нельзя вызывать подпрограммы MPI. MPI_Finalize должны вызывать все процессы перед завершением своей работы.

int MPI_Comm_size(MPI_Comm comm, int *size)

Определение размера области взаимодействия.

Входные параметры:

- comm - коммуникатор.

Выходные параметры:

- size - количество процессов в области взаимодействия.

int MPI_Comm_rank(MPI_Comm comm, int *rank)

Определение ранга процесса.

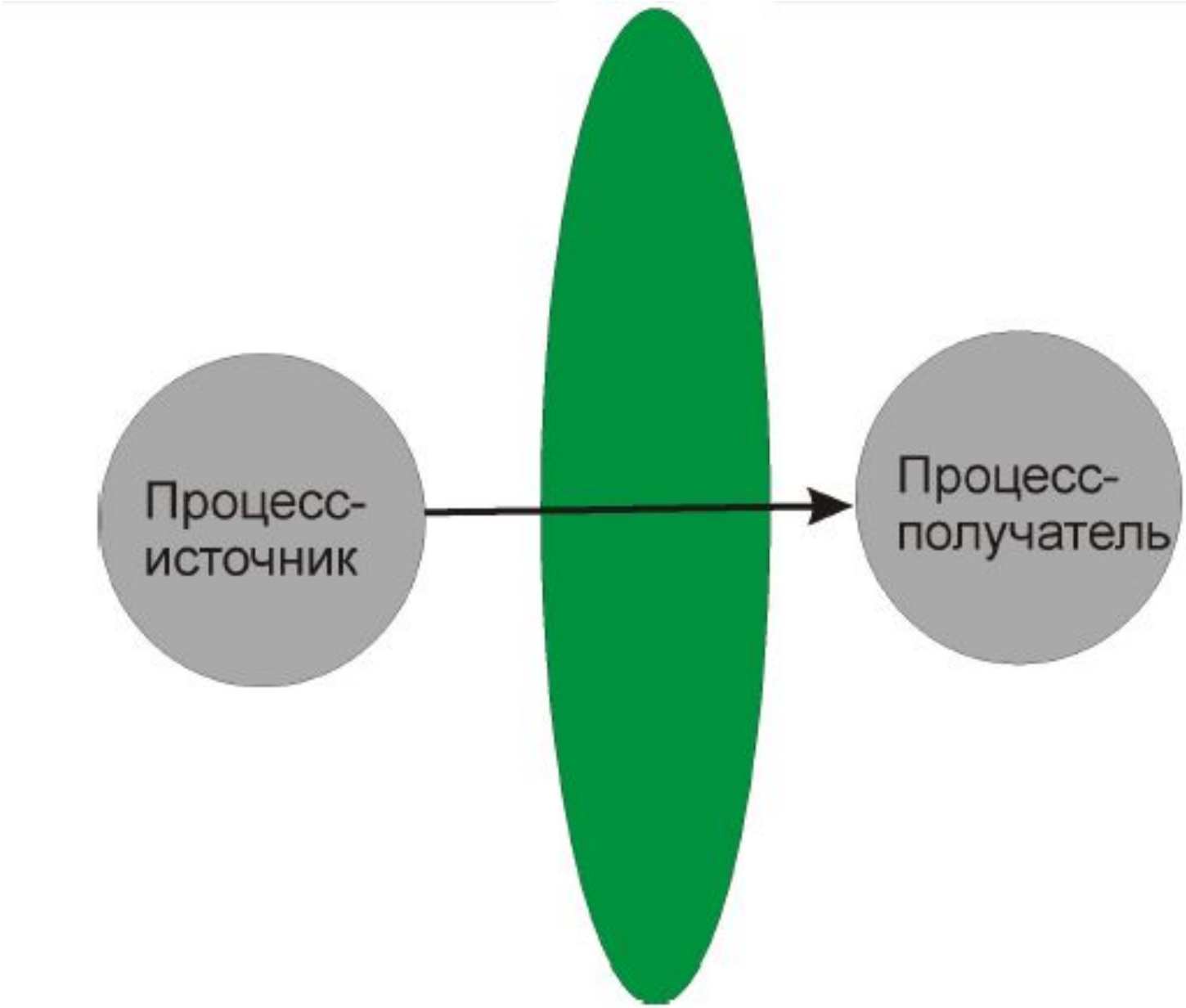
Входные параметры:

- comm - коммуникатор.

Выходные параметры:

- rank - ранг процесса в области взаимодействия.

Двухточечный обмен



Стандартная блокирующая передача

int MPI_Send(void *buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm)

Входные параметры:

- buf - адрес первого элемента в буфере передачи
- count - количество элементов в буфере передачи;
- datatype - тип MPI каждого пересылаемого элемента;
- dest - ранг процесса-получателя сообщения (целое число от 0 до n – 1, где n ¾ число процессов в области взаимодействия);
- tag - тег сообщения;
- comm - коммуникатор;

Возвращает код завершения.

Стандартный блокирующий прием

int MPI_Recv(void *buf, int count, MPI_Datatype datatype, int source, int tag, MPI_Comm comm, MPI_Status *status)

Входные параметры:

- count - максимальное количество элементов в буфере приема. Фактическое их количество можно определить с помощью подпрограммы MPI_Get_count;
- datatype - тип принимаемых данных. Напомним о необходимости соблюдения соответствия типов аргументов подпрограмм приема и передачи;
- source - ранг источника. Можно использовать специальное значение MPI_ANY_SOURCE, соответствующее произвольному значению ранга. В программировании идентификатор, отвечающий произвольному значению параметра, часто называют "джокером". Этот термин будем использовать и мы;
- tag - тег сообщения или "джокер" MPI_ANY_TAG, соответствующий произвольному значению тега;
- comm - коммуникатор. При указании коммуникатора "джокеры" использовать нельзя.

Выходные параметры:

- buf - начальный адрес буфера приема. Его размер должен быть достаточным, чтобы разместить принимаемое сообщение, иначе при выполнении приема произойдет сбой возникнет ошибка переполнения;
- status - статус обмена.

Если сообщение меньше, чем буфер приема, изменяется содержимое лишь тех ячеек памяти буфера, которые относятся к сообщению.

Send-Recv example

```
#include <mpi.h>
int main(int argc, char* argv[]){
	int rank, buf[256];
	MPI_Status status;
	MPI_Init(&argc, &argv);
	MPI_Comm_rank(MPI_COMM_WORLD, &rank);
	if(rank == 0){
		// init buffer
		MPI_Send(buf, 256, MPI_INT, 1, 10, MPI_COMM_WORLD);
	}
	if(rank == 1){
		MPI_Recv(buf, 256, MPI_INT, 0, 10, MPI_COMM_WORLD, &status);
		// process buffer
	}
	MPI_Finalize();
	return 0;
}
```

Wildcard

- Получить сообщение от любого источника —

source = MPI_ANY_SOURCE

- Получить сообщение с любым tag —

tag = MPI_ANY_TAG

- Реальные значения source и tag возвращаются в структуре status.

Определение размера полученного сообщения (count)

int MPI_Get_count(MPI_Status *status, MPI_Datatype datatype, int*count)

Аргумент datatype должен соответствовать типу данных, указанному в операции передачи сообщения.

Буферизованный обмен

int MPI_Bsend(void *buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm)

Параметры совпадают с параметрами подпрограммы MPI_Send.

Создание буфера

int MPI_Buffer_attach(void *buf, size)

Выходной параметр:

• buf - буфер размером size байтов.

Роль буфера может играть массив. Этот массив должен быть описан в программе, его не следует использовать для других целей (например, в качестве первого аргумента подпрограммы MPI_Bsend). За один раз к процессу может быть подключен только один буфер.

Отключение буфера

int MPI_Buffer_detach(void *buf, int *size)

Выходные параметры:

• buf - адрес;

• size - размер отключаемого буфера.

Вызов данной подпрограммы блокирует работу процесса до тех пор, пока все сообщения, находящиеся в буфере, не будут обработаны. В языке С данный вызов не освобождает автоматически память, отведенную для буфера.

Прием и передача данных с блокировкой

int MPI_Sendrecv(void *sendbuf, int sendcount, MPI_Datatype sendtype, int dest, int sendtag, void *recvbuf, int recvcount, MPI_Datatype recvtype, int source, int recvtag, MPI_Comm comm, MPI_Status *status)

Входные параметры:

- sendbuf - начальный адрес буфера передачи;
- sendcount - количество передаваемых элементов;
- sendtype - тип передаваемых элементов;
- dest - ранг адресата;
- sendtag - тег передаваемого сообщения;
- recvbuf - начальный адрес буфера приема;
- recvcount - количество элементов в буфере приема;
- recvtype - тип элементов в буфере приема;
- source - ранг источника;
- recvtag - тег принимаемого сообщения;
- comm - коммуникатор.

Выходные параметры:

- recvbuf - начальный адрес буфера приема;
- status - статус операции приема.

Прием, и передача используют один и тот же коммуникатор. Буферы передачи и приема не должны пересекаться, у них может быть разный размер, типы пересылаемых и принимаемых данных также могут различаться.

Неблокирующие операции

Выполнение в три этапа:

- Инициализация
  - немедленный выход
  - функции начинаются с MPI_I…
- Выполняем какую-то работу
- Ожидание завершения

int MPI_Isend(void *buf, int count, MPI_Datatype datatype, int dest, int msgtag, MPI_Comm comm, MPI_Request *request)

- *buf* - адрес начала буфера посылки сообщения
- *count* - число передаваемых элементов в сообщении
- *datatype* - тип передаваемых элементов
- *dest* - номер процесса-получателя
- *msgtag* - идентификатор сообщения
- *comm* - идентификатор группы
- OUT *request* - идентификатор асинхронной передачи

Передача сообщения, аналогичная *MPI_Send*, однако возврат из подпрограммы происходит сразу после инициализации процесса передачи без ожидания обработки всего сообщения, находящегося в буфере *buf*. Это означает, что нельзя повторно использовать данный буфер для других целей без получения дополнительной информации о завершении данной посылки. Окончание процесса передачи (т.е. того момента, когда можно переиспользовать буфер *buf*без опасения испортить передаваемое сообщение) можно определить с помощью параметра *request* и процедур *MPI_Wait* и *MPI_Test*. Сообщение, отправленное любой из процедур *MPI_Send* и *MPI_Isend*, может быть принято любой из процедур *MPI_Recv* и *MPI_Irecv*.

int MPI_Irecv(void *buf, int count, MPI_Datatype datatype, int source, int msgtag, MPI_Comm comm, MPI_Request *request)

- OUT *buf* - адрес начала буфера приема сообщения
- *count* - максимальное число элементов в принимаемом сообщении
- *datatype* - тип элементов принимаемого сообщения
- *source* - номер процесса-отправителя
- *msgtag* - идентификатор принимаемого сообщения
- *comm* - идентификатор группы
- OUT *request* - идентификатор асинхронного приема сообщения

Прием сообщения, аналогичный *MPI_Recv*, однако возврат из подпрограммы происходит сразу после инициализации процесса приема без ожидания получения сообщения в буфере *buf*. Окончание процесса приема можно определить с помощью параметра *request* и функций *MPI_Wait* и *MPI_Test*.

int MPI_Wait(MPI_Request *request, MPI_Status *status)

- *request* - идентификатор асинхронного приема или передачи
- OUT *status* - параметры сообщения
- Ожидание завершения асинхронных процедур *MPI_Isend* или *MPI_Irecv*, ассоциированных с идентификатором *request*. В случае приема, атрибуты и длину полученного сообщения можно определить обычным образом с помощью параметра *status*.

int MPI_Waitall(int count, MPI_Request *requests, MPI_Status *statuses)

int MPI_Waitany(int count, MPI_Request *requests, int *index, MPI_Status *status)

- *count* - число идентификаторов
- *requests* - массив идентификаторов асинхронного приема или передачи
- OUT *index* - номер завершенной операции обмена
- OUT *status(es)* - параметры сообщений

int MPI_Waitsome(int incount, MPI_Request *requests, int *outcount, int *indexes, MPI_Status *statuses)

- *incount* - число идентификаторов
- *requests* - массив идентификаторов асинхронного приема или передачи
- OUT *outcount* - число идентификаторов завершившихся операций обмена
- OUT *indexes* - массив номеров завершившихся операции обмена
- OUT *statuses* - параметры завершившихся сообщений
- Выполнение процесса блокируется до тех пор, пока по крайней мере одна из операций обмена, ассоциированных с указанными идентификаторами, не будет завершена. Параметр *outcount* содержит число завершенных операций, а первые *outcount* элементов массива *indexes* содержат номера элементов массива *requests* с их идентификаторами. Первые *outcount* элементов массива *statuses* содержат параметры завершенных операций.

- **int MPI_Test(MPI_Request *request, int *flag, MPI_Status *status)**
- *request* - идентификатор асинхронного приема или передачи
- OUT *flag* - признак завершенности операции обмена
- OUT *status* - параметры сообщения
- Проверка завершенности асинхронных процедур *MPI_Isend* или *MPI_Irecv*, ассоциированных с идентификатором *request*. В параметре *flag* возвращает значение 1, если соответствующая операция завершена, и значение 0 в противном случае. Если завершена процедура приема, то атрибуты и длину полученного сообщения можно определить обычным образом с помощью параметра *status*.
- **int MPI_Testall(int count, MPI_Request *requests, int *flag, MPI_Status *statuses)**
- **int MPI_Testany(int count, MPI_Request *requests, int *index, int *flag, MPI_Status *status)**

```
#include "mpi.h“
#include <stdio.h>
    int main(int argc, char *argv[])
    {
        int myid, numprocs, left, right;
        int buffer[10], buffer2[10];
        MPI_Request request, request2;
        MPI_Status status;

        MPI_Init(&argc,&argv);
        MPI_Comm_size(MPI_COMM_WORLD, &numprocs);
        MPI_Comm_rank(MPI_COMM_WORLD, &myid);
        right = (myid + 1) % numprocs;
        left = myid - 1;
        if (left < 0)
            left = numprocs - 1;
        MPI_Irecv(buffer, 10, MPI_INT, left, 123, MPI_COMM_WORLD,
    &request);
        MPI_Isend(buffer2, 10, MPI_INT, right, 123, MPI_COMM_WORLD,
    &request2);
        MPI_Wait(&request, &status);
        MPI_Wait(&request2, &status);
        MPI_Finalize();
        return 0;
    }
```

Коллективные операции

Взаимодействует группа процессов.

● Вовлечены все процессы в коммуникаторе

● Примеры:

– Синхронизация (барьер).

– Broadcast, scatter, gather.

– Global sum, global maximum, etc.

Характеристики:

Коллективная над коммуникатором

● ВСЕ процессы в рамках коммуникатора должны вызвать коллективную функцию

● Коллективные операции - блокирующие.

● Нет tag-а.

● Буфер на отправку данных должен быть такого же размера как и буфер на прием данных.

Barrier Synchronization

- C: int MPI_Barrier(MPI_Comm comm)
- MPI_Barrier обычно не используется, но:

– Используется для отладки:

- Не забывайте удалять после отладки.

– Используется для профилирования

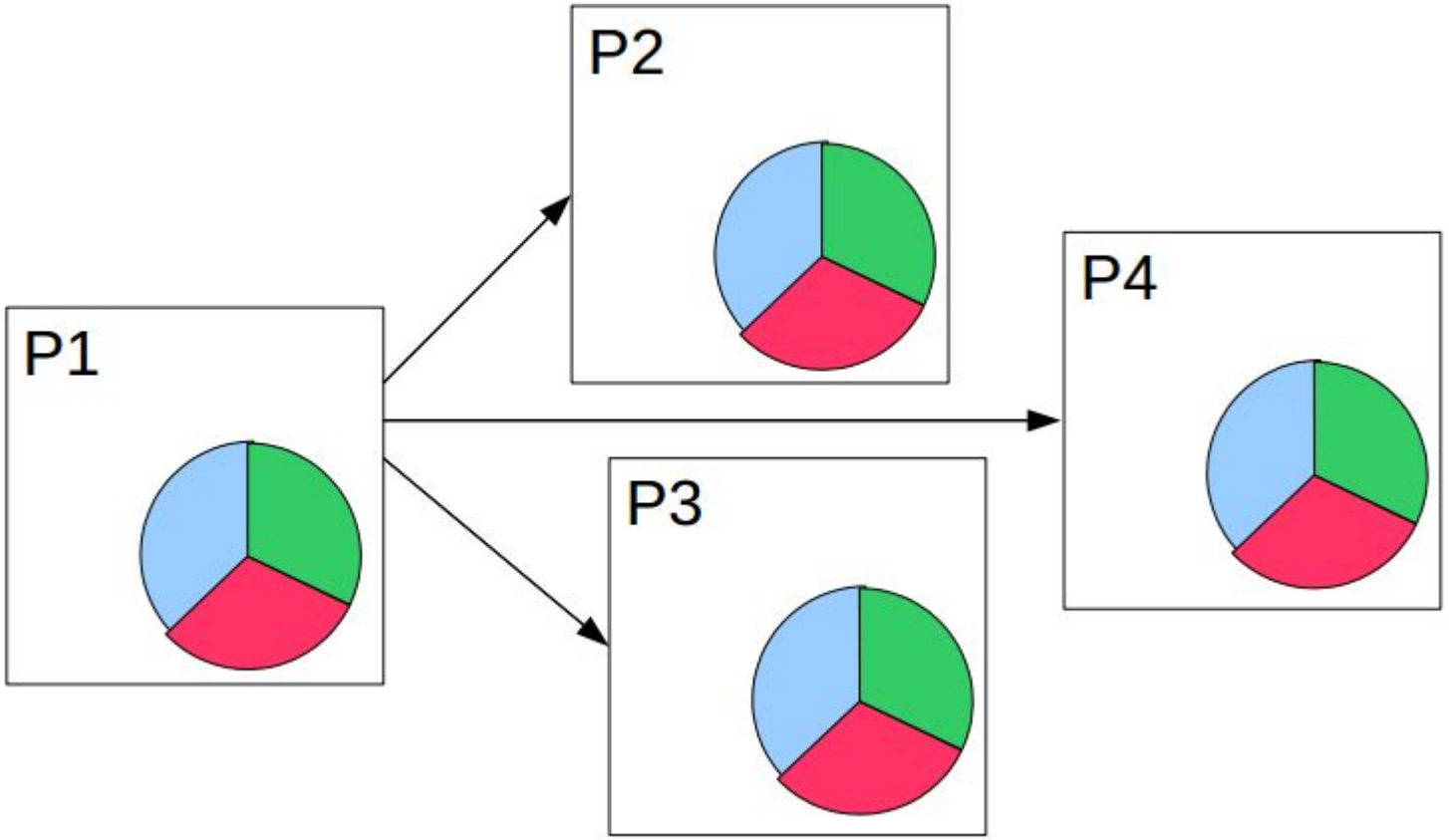
- Load imbalance of computation [MPI_Wtime(); MPI_Barrier(); .... MPI_Wtime()]
- communication epochs [MPI_Wtime(); MPI_Allreduce(); …; MPI_Wtime()]

– Используется для синхронизации внешних операций (например, I/O):

- Посылка сообщений может быть более эффективна

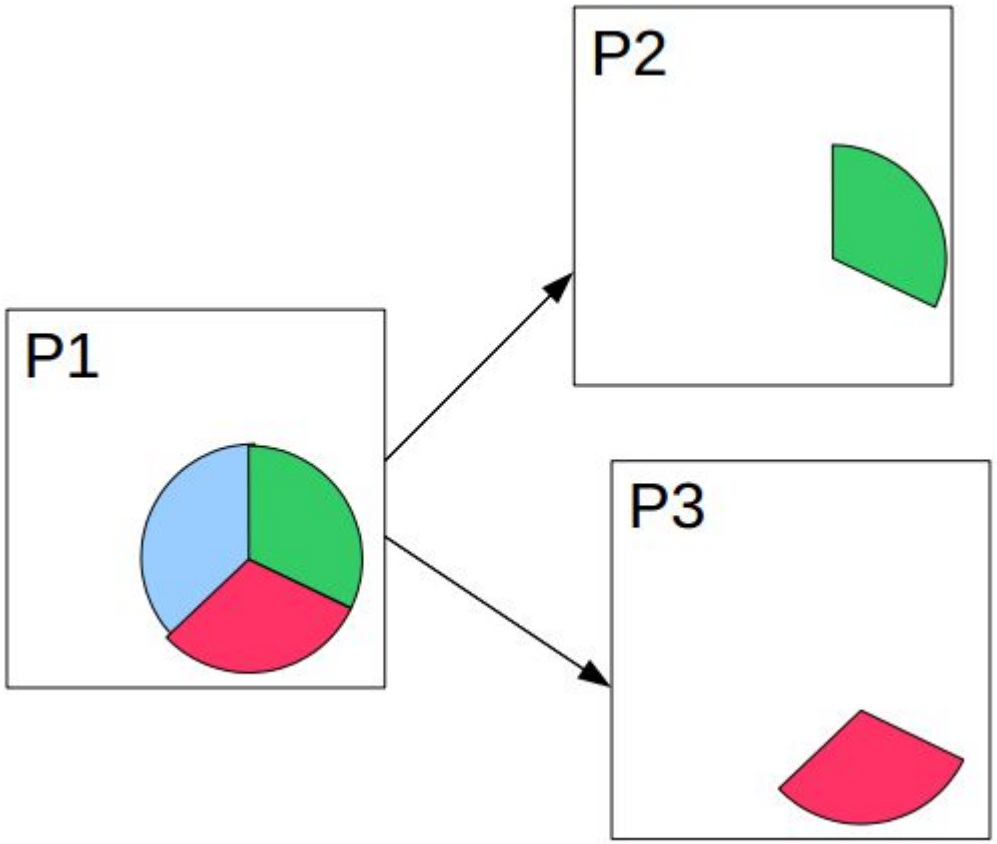
- **Broadcast**

int MPI_Bcast(void *buf, int count, PI_Datatype datatype,int root, MPI_Comm comm)



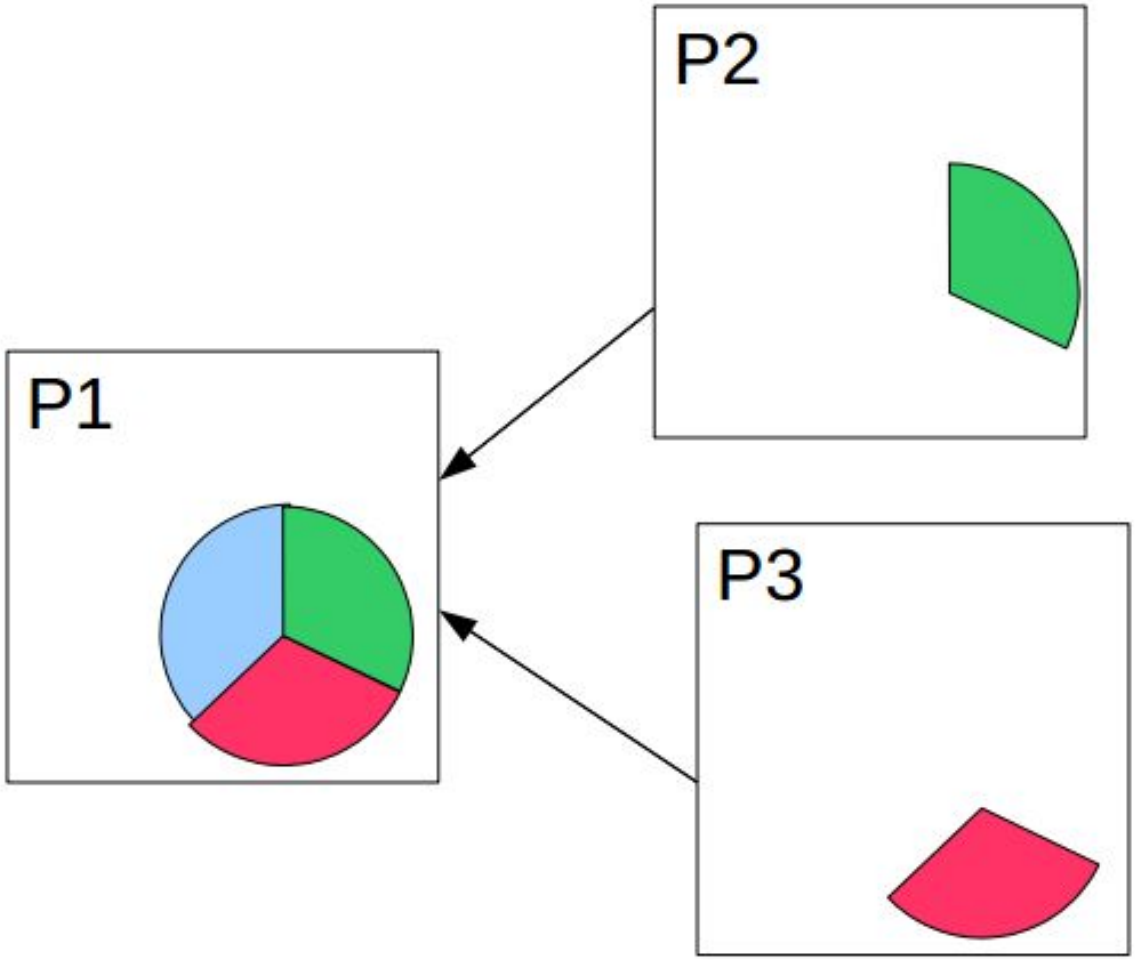
- **Scatter**

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



- **Gather**

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



Глобальные операции редукции

- Сумма
- Произведение
- Минимум
- Максимум
- Битовые операции

«Собираемая» переменная может быть простой или массивом.

Операторы редукции

Операторы	Функции
MPI_BAND	Bitwise AND
MPI_BOR	Bitwise OR
MPI_BXOR	Bitwise exclusive OR
MPI_MAX	Maximum
MPI_MAXLOC	Maximum and location of the maximum
MPI_MIN	Minimum
MPI_MINLOC	Minimum and location of the minimum
MPI_LAND	Logical AND
MPI_LOR	Logical OR
MPI_LXOR	Logical exclusive OR
MPI_PROD	Product
MPI_SUM	Sum

Операция приведения, результат которой передается одному процессу

int MPI_Reduce(void *buf, void *result, int count, MPI_Datatype datatype, MPI_Op op, int root, MPI_Comm comm)

Входные параметры:

- buf - адрес буфера передачи;
- count - количество элементов в буфере передачи;
- datatype - тип данных в буфере передачи;
- op - операция приведения;
- root - ранг главного процесса;
- comm - коммуникатор.

MPI_Reduce применяет операцию приведения к операндам из buf, а результат каждой операции помещается в буфер результата result. MPI_Reduce должна вызываться всеми процессами в коммуникаторе comm, а аргументы count, datatype и op в этих вызовах должны совпадать.

Определение собственных глобальных операций

int MPI_Op_create(MPI_User_function *function, int commute, MPI_Op *op)

Входные параметры:

- function - пользовательская функция;
- commute - флаг, которому присваивается значение "истина", если операция коммутативна (результат не зависит от порядка операндов).

Описание типа пользовательской функции выглядит следующим образом:

typedef void (MPI_User_function)(void *a, void *b, int *len, MPI_Datatype *dtype)

Здесь операция определяется так:

b[I] = a[I] op b[I]

для I = 0, …, len – 1.

```
int res, gres;
char * buf, *rbuf;
MPI_Init(&argc, &argv);
// init bufs
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Comm_size(MPI_COMM_WORLD, &size);
while(/* have data */){
    if(rank == 0) read_data(buf, 100*size);
    MPI_Scatter(buf, 100, MPI_CHAR, rbuf, 100,
            MPI_CHAR, 0, MPI_COMM_WORLD);
    res = process_data(rbuf, 100);
    MPI_Reduce(&res, &gres, 1, MPI_INT, MPI_XOR, 0,
MPI_COMM_WORLD);
save_res(gres);
}
```