# Парные операции приема-передачи сообщений

**4.1.2. Операции передачи данных**

Основу MPI составляют операции передачи сообщений. Среди предусмотренных в составе MPI функций различаются парные (point-to-point) операции между двумя процессами и коллективные (collective) коммуникационные действия для одновременного взаимодействия нескольких процессов.

Для выполнения парных операций могут использоваться разные режимы передачи, среди которых синхронный, блокирующий и др. – полное рассмотрение возможных режимов передачи будет выполнено в подразделе 4.3.

Как уже отмечалось ранее, стандарт MPI предусматривает необходимость реализации большинства основных коллективных операций передачи данных – см. подразделы 4.2 и 4.4.

**4.1.3. Понятие коммуникаторов**

Процессы параллельной программы объединяются в группы. 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 3 коммуникатора. Как результат, указание используемого коммуникатора является обязательным для операций передачи данных в MPI.

В ходе вычислений могут создаваться новые и удаляться существующие группы процессов и коммуникаторы. Один и тот же процесс может принадлежать разным группам и коммуникаторам. 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).

**Передача сообщений**

Для передачи сообщения процесс-отправитель должен выполнить функцию:

Изображение выглядит как текст

Автоматически созданное описание

Для указания типа пересылаемых данных в MPI имеется ряд базовых типов, полный список которых приведен в табл. 4.1.

Изображение выглядит как стол

Автоматически созданное описание

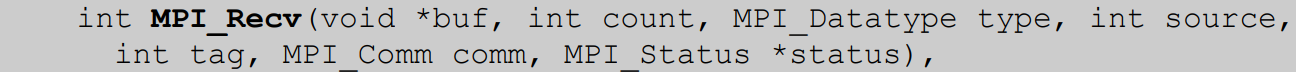
Следует отметить:

1. Отправляемое сообщение определяется через указание блока памяти (буфера), в котором это сообщение располагается. Используемая для указания буфера триада ( buf, count, type ) входит в состав параметров практически всех функций передачи данных,
2. Процессы, между которыми выполняется передача данных, в обязательном порядке должны принадлежать коммуникатору, указываемому в функции MPI\_Send,
3. Параметр tag используется только при необходимости различения передаваемых сообщений, в противном случае в качестве значения параметра может быть использовано произвольное целое число (см. также описание функции MPI\_Recv).

Сразу же после завершения функции MPI\_Send процесс-отправитель может начать повторно использовать буфер памяти, в котором располагалось отправляемое сообщение. Вместе с этим, следует понимать, что в момент завершения функции MPI\_Send состояние самого пересылаемого сообщения может быть совершенно различным - сообщение может располагаться в процессе-отправителе, может находиться в процессе передачи, может храниться в процессе-получателе или же может быть принято процессомполучателем при помощи функции MPI\_Recv. Тем самым, завершение функции MPI\_Send означает лишь, что операция передачи начала выполняться и пересылка сообщения будет рано или поздно будет выполнена.

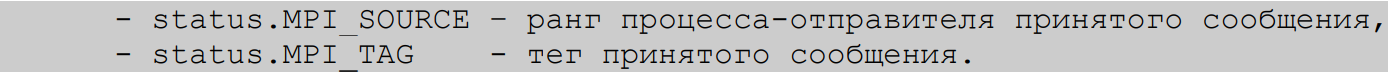
**4.2.1.4 Прием сообщений**

Для приема сообщения процесс-получатель должен выполнить функцию: int

  
Изображение выглядит как текст

Автоматически созданное описание

Следует отметить:

1. Буфер памяти должен быть достаточным для приема сообщения, а тип элементов передаваемого и принимаемого сообщения должны совпадать; при нехватке памяти часть сообщения будет потеряна и в коде завершения функции будет зафиксирована ошибка переполнения,
2. При необходимости приема сообщения от любого процесса-отправителя для параметра source может быть указано значение MPI\_ANY\_SOURCE,
3. При необходимости приема сообщения с любым тегом для параметра tag может быть указано значение MPI\_ANY\_TAG,
4. Параметр status позволяет определить ряд характеристик принятого сообщения

Функция 

возвращает в переменной count количество элементов типа type в принятом сообщении.

Вызов функции MPI\_Recv не должен согласовываться со временем вызова соответствующей функции передачи сообщения MPI\_Send – прием сообщения может быть инициирован до момента, в момент или после момента начала отправки сообщения.

По завершении функции MPI\_Recv в заданном буфере памяти будет располагаться принятое сообщение. Принципиальный момент здесь состоит в том, что функция MPI\_Recv является блокирующей для процесса-получателя, т.е. его выполнение приостанавливается до завершения работы функции. Таким образом, если по каким-то причинам ожидаемое для приема сообщение будет отсутствовать, выполнение параллельной программы будет блокировано.

# Передача данных: синхронная, по готовности, буферизованная

**4.3.1. Режимы передачи данных**

Рассмотренная ранее функция MPI\_Send обеспечивает так называемый стандартный (Standard) режим отправки сообщений, при котором (см. также п. 4.2.1.3):

* на время выполнения функции процесс-отправитель сообщения блокируется,
* после завершения функции буфер может быть использован повторно,
* состояние отправленного сообщения может быть различным - сообщение может располагаться в процессе-отправителе, может находиться в процессе передачи, может храниться в процессе-получателе или же может быть принято процессом-получателем при помощи функции MPI\_Recv.

Кроме стандартного режима в MPI предусматриваются следующие дополнительные режимы передачи сообщений:

* Синхронный (Synchronous) режим состоит в том, что завершение функции отправки сообщения происходит только при получении от процесса-получателя подтверждения о начале приема отправленного сообщения, отправленное сообщение или полностью принято процессом-получателем или находится в состоянии приема,
* Буферизованный (Buffered) режим предполагает использование дополнительных системных буферов для копирования в них отправляемых сообщений; как результат, функция отправки сообщения завершается сразу же после копирования сообщения в системный буфер,
* Режим передачи по готовности (Ready) может быть использован только, если операция приема сообщения уже инициирована. Буфер сообщения после завершения функции отправки сообщения может быть повторно использован.

Для именования функций отправки сообщения для разных режимов выполнения в MPI используется название функции MPI\_Send, к которому как префикс добавляется начальный символ названия соответствующего режима работы, т.е.

* **MPI\_Ssend** – функция отправки сообщения в синхронном режиме,
* **MPI\_Bsend** – функция отправки сообщения в буферизованном режиме,
* **MPI\_Rsend** – функция отправки сообщения в режиме по готовности.

Список параметров всех перечисленных функций совпадает с составом параметров функции MPI\_Send.

Для использования буферизованного режима передачи должен быть создан и передан MPI буфер памяти для буферизации сообщений – используемая для этого функция имеет вид:

Изображение выглядит как текст

Автоматически созданное описание

После завершения работы с буфером он должен быть отключен от MPI при помощи функции:



По практическому использованию режимов можно привести следующие рекомендации:

1. Режим передачи по готовности формально является наиболее быстрым, но используется достаточно редко, т.к. обычно сложно гарантировать готовность операции приема,
2. Стандартный и буферизованный режимы также выполняются достаточно быстро, но могут приводить к большим расходам ресурсов (памяти) – в целом может быть рекомендован для передачи коротких сообщений,
3. Синхронный режим является наиболее медленным, т.к. требует подтверждения приема. В тоже время, этот режим наиболее надежен – можно рекомендовать его для передачи длинных сообщений.

В заключение отметим, что для функции приема MPI\_Recv не существует различных режимов работы.

# Совмещение отправки и приёма

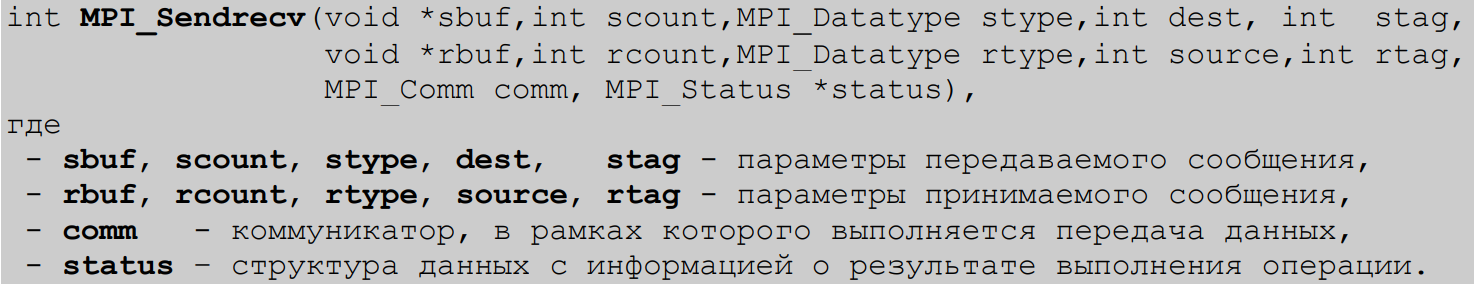
**4.3.3. Одновременное выполнение передачи и приема**

Одной из часто выполняемых форм информационного взаимодействия в параллельных программах является обмен данными между процессами, когда для продолжения вычислений процессам необходимо отправить данные одним процессам и, в то же время, получить сообщения от других процессов.

Простейший вариант этой ситуации состоит, например, в обмене данными между двумя процессами.

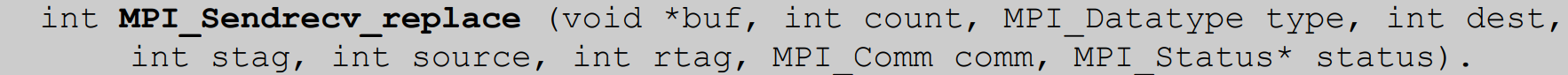
Реализация таких обменов при помощи обычных парных операций передачи данных неэффективна и достаточно трудоемка. Кроме того, такая реализация должна гарантировать отсутствие тупиковых ситуаций, которые могут возникать, например, когда два процесса начинают передавать сообщения друг другу с использованием блокирующих функций передачи данных.

Достижение эффективного и гарантированного одновременного выполнения операций передачи и приема данных может быть обеспечено при помощи функции MPI:



Как следует из описания, функция MPI\_Sendrecv передает сообщение, описываемое параметрами (sbuf, scount, stype, dest, stag), процессу с рангом dest и принимает сообщение в буфер, определяемый параметрами (rbuf, rcount, rtype, source, rtag), от процесса с рангом source.

В функции MPI\_Sendrecv для передачи и приема сообщений применяются разные буфера. В случае же, когда сообщения имеют одинаковый тип, в MPI имеется возможность использования единого буфера:

Пример использование функций для одновременного выполнения операций передачи и приема приведен в разделе 8 при разработке параллельных программ матричного умножения.

# Проверка сообщения без приема (MPI\_Probe).

**Проверка состояния**

**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 – 1, если соответствующая операция завершена, и значение 0 в противном случае. Если завершена процедура приема, то атрибуты и длину полученного сообщения можно определить с помощью параметра status.

**int MPI\_Iprobe( int source, int msgtag, MPI\_Comm comm, int \*flag, MPI\_Status \*status)**

**source** – номер процесса-отправителя или MPI\_ANY\_SOURCE;   
**msgtag** – идентификатор ожидаемого сообщения или MPI\_ANY\_TAG;   
**OUT status** – параметры обнаруженного сообщения.

Получение информации в массиве status о поступлении и структуре ожидаемого сообщения без блокировки. В параметре flag значение 1, если сообщение с подходящими атрибутами уже может быть принято (в этом случае ее действие полностью аналогично MPI\_Probe), и значение 0, если сообщения с указанными атрибутами еще нет.

**Пример MPI\_Probe**

Процедура MPI\_Probe использована для определения структуры приходящего сообщения. Процесс 0 ждет сообщения от любого из процессов 1 и 2 с одинаковым тегом. Посылаемые данные имеют разный тип.

Чтобы определить в какую переменную помещать приходящее сообщение, процесс определяет от кого оно поступило.

Следующий после MPI\_Probe вызов MPI\_Recv гарантировано примет нужное сообщение, далее принимается сообщение от другого процесса.