
Коммуникационные операции «точка-точка»

параллельное программирование

Коммуникационные операции типа «точка-точка»

К операциям этого типа относятся две представленные выше коммуникационные процедуры (MPI_Send, MPI_Recv). В коммуникационных операциях типа точка-точка всегда участвуют 2 процесса: передающий и принимающий. В MPI имеется множество функций, реализующих такой тип обмена. Многообразие объясняется возможностью организации таких обменов множеством способов. Описанные в предыдущем разделе функции реализуют стандартный режим с блокировкой.

Блокирующие функции

- *Блокирующие функции* подразумевают полное окончание операции после выхода из процедуры, т.е. вызывающий процесс блокируется, пока операция не будет завершена.

Неблокирующие функции

- *Неблокирующие функции* подразумевают совмещение операций обмена с другими операциями, поэтому неблокирующие функции передачи и приема по сути дела являются функциями инициализации соответствующих операций. Для опроса завершенности операции (и принудительного завершения) вводятся дополнительные функции.

Коммуникационные операции типа «Точка-Точка»

| Способ связи | С блокировкой | Без блокировки |
|--------------------------|---------------|----------------|
| Стандартная посылка | MPI_Send | MPI_Isend |
| Синхронная посылка | MPI_Ssend | MPI_Issend |
| Буферизированная посылка | MPI_Bsend | MPI_Ibsend |
| Согласованная посылка | MPI_Rsend | MPI_Irsend |
| Прием информации | MPI_Recv | MPI_Irecv |

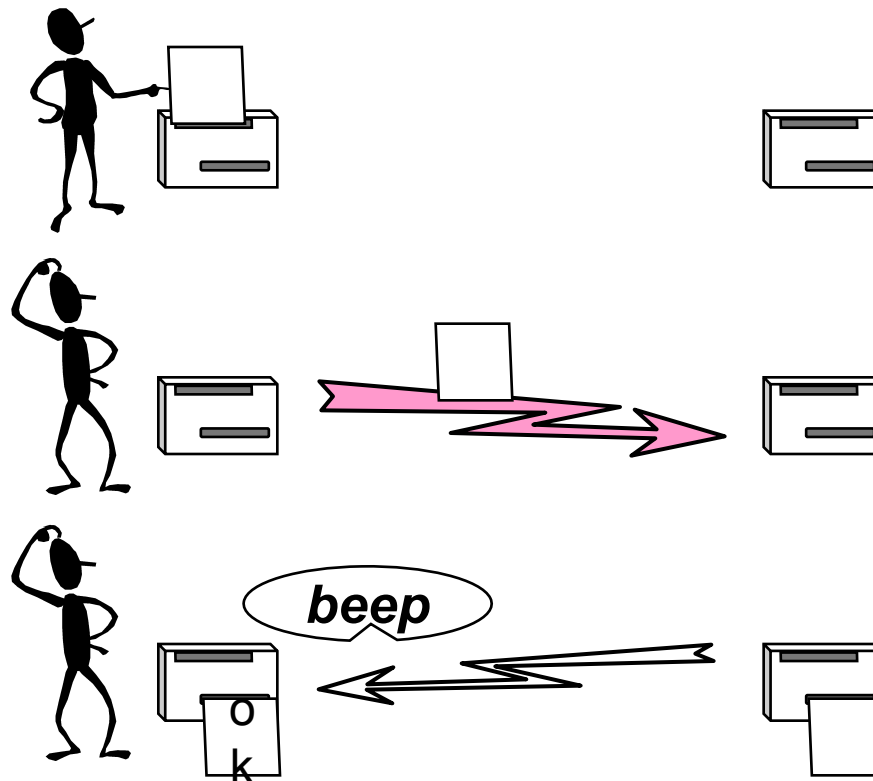
- Из таблицы хорошо виден принцип формирования имен функций. К именам базовых функций Send/Recv добавляются различные префиксы.

Коммуникационные операции типа «точка-точка»

| | |
|-----------|--|
| Префикс S | (synchronous) – означает синхронный режим передачи данных. Операция передачи данных заканчивается только тогда, когда заканчивается прием данных. |
| Префикс B | (buffered) – означает буферизованный режим передачи данных. В адресном пространстве передающего процесса с помощью специальной функции (MPI_Buffer_attach) создается буфер обмена, который используется в операциях обмена. Операция отправки заканчивается, когда данные помещены в этот буфер. |
| Префикс R | (ready) – согласованный или подготовленный режим передачи данных. Операция передачи данных должна начинаться только тогда, когда принимающий процессор уже точно выставил признак готовности приема данных. |
| Префикс I | (immediate) – относится к неблокирующим операциям. |

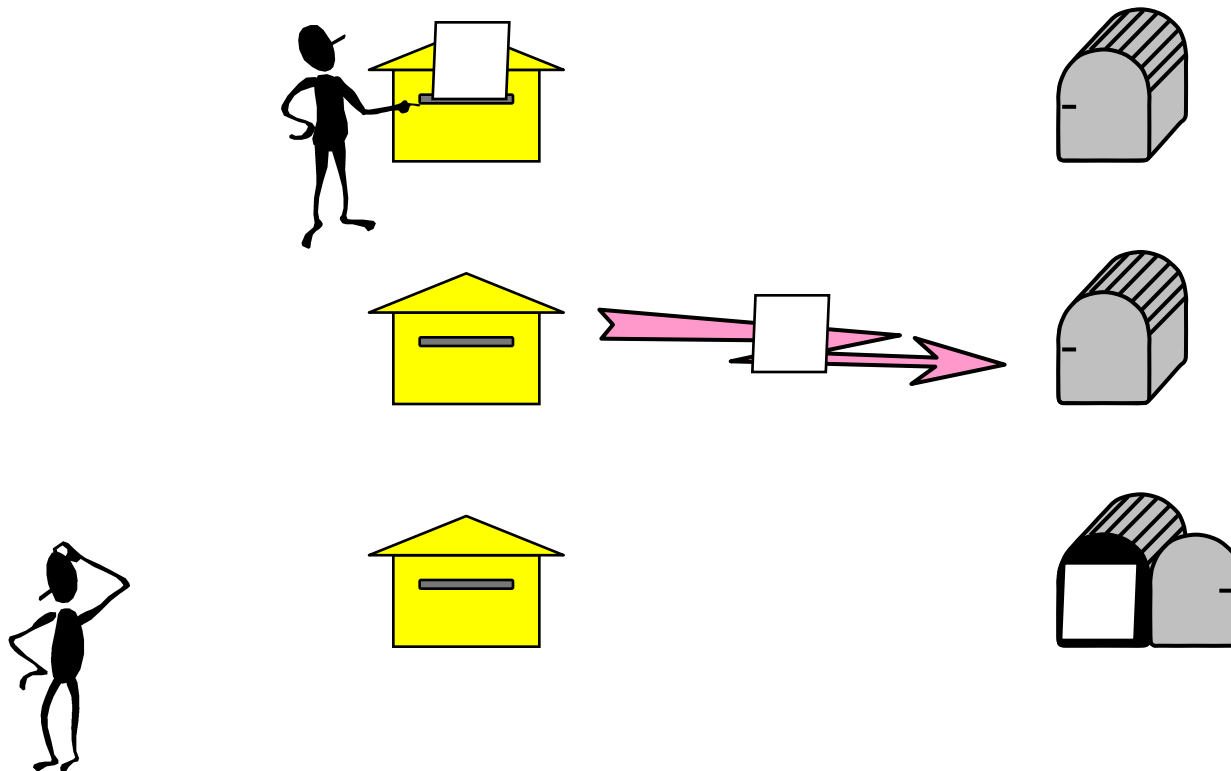
MPI – Синхронная посылка

- Процессор-отправитель ожидает информацию о том, когда получатель примет сообщение.
- Пример, факс получатель присылает тег завершения приема.



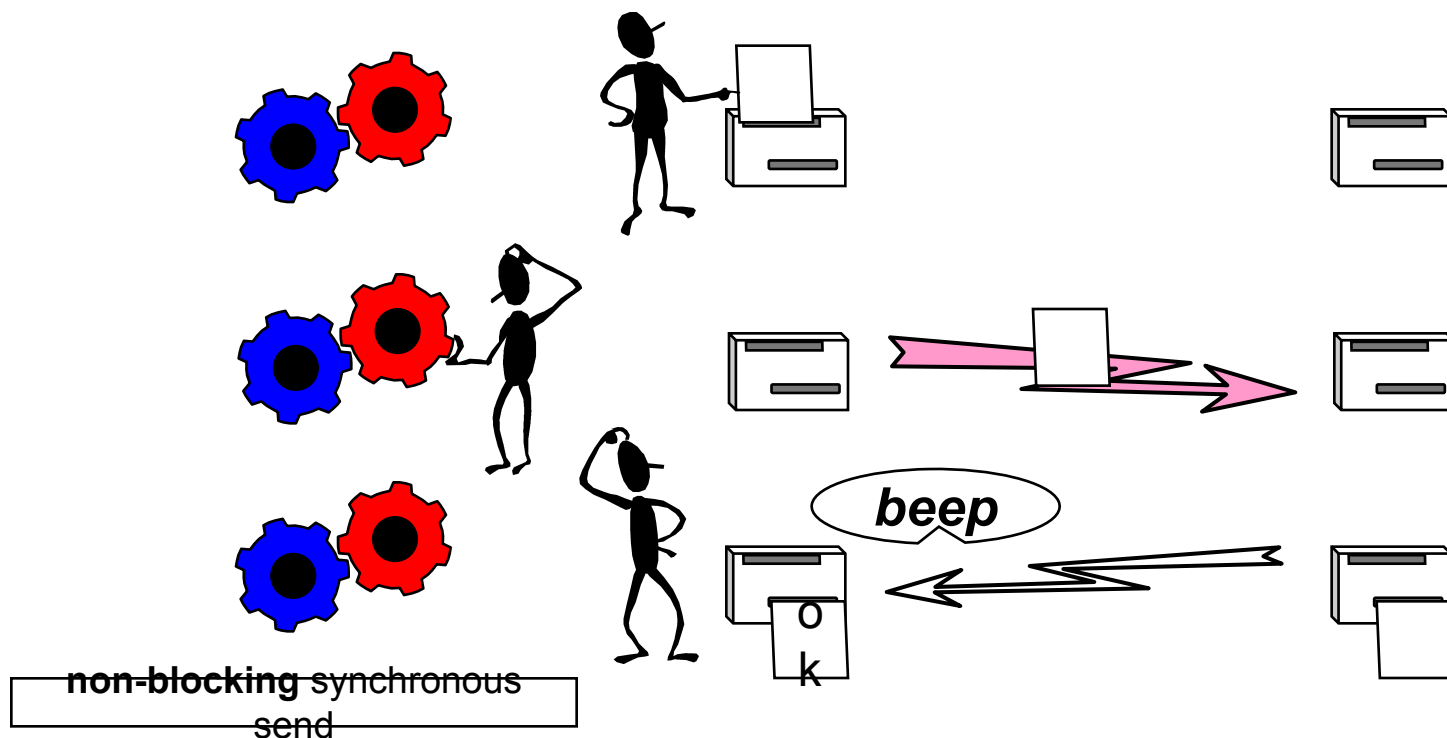
МРІ – Буферизированная посылка или Несинхронная посылка

- Процессор-отправитель знает только когда сообщение ушло.



MPI–посылки без блокировки

- Неблокирующие операции немедленно возвращают управление программе. Программа выполняет следующие действия.
- Для того, что бы спустя некоторое время убедиться, что неблокирующая функция передачи данных выполнена полностью, нужно вызвать функцию `MPI_Test` или `MPI_Wait`.



Неблокирующие коммуникационные операции

- Использование неблокирующих коммуникационных операций более безопасно с точки зрения возникновения тупиковых ситуаций, а также может увеличить скорость работы программы за счет совмещения выполнения вычислительных и коммуникационных операций. Эти задачи решаются разделением коммуникационных операций на две стадии: инициирование операции и проверку завершения операции.

Неблокирующие коммуникационные операции

- Неблокирующие операции используют специальный скрытый (opaque) объект "запрос обмена" (request) для связи между функциями обмена и функциями опроса их завершения.
- Для прикладных программ доступ к этому объекту возможен только через вызовы MPI-функций.
- Если операция обмена завершена, подпрограмма проверки снимает "запрос обмена", устанавливая его в значение `MPI_REQUEST_NULL`.
- Снять "запрос обмена" без ожидания завершения операции можно подпрограммой `MPI_Request_free`.

Функция передачи сообщений MPI_Isend

```
int MPI_Isend( void *sbuf, int count, MPI_Datatype
datatype, int dest, int tag, MPI_Comm comm,
MPI_Request *request )
```

■ Входные параметры:

| | |
|-----------------------|--|
| <code>sbuf</code> | адрес начала расположения пересылаемых данных |
| <code>count</code> | число пересылаемых элементов |
| <code>datatype</code> | тип посылаемых элементов |
| <code>dest</code> | номер процесса-получателя в группе, связанной с коммуникатором <code>comm</code> |
| <code>tag</code> | идентификатор сообщения |
| <code>comm</code> | коммуникатор области связи |

■ Выходные параметры:

| | |
|----------------------|---------------|
| <code>request</code> | запрос обмена |
|----------------------|---------------|

Функция приема сообщений MPI_Irecv

```
int MPI_Irecv( void *rbuf, int count, MPI_Datatype  
datatype, int source, int tag, MPI_Comm comm,  
MPI_Request *request )
```

■ Входные параметры:

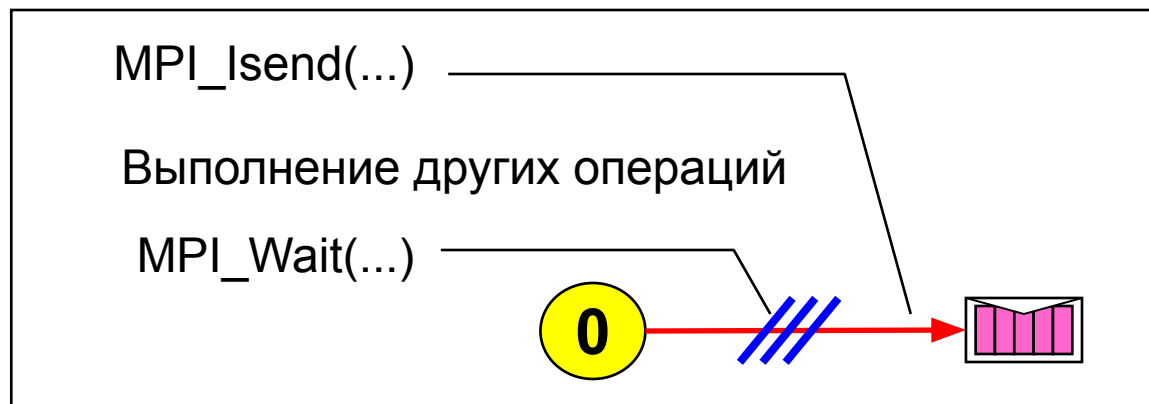
| | |
|-----------------------|------------------------------|
| <code>count</code> | число пересылаемых элементов |
| <code>datatype</code> | тип посылаемых элементов |
| <code>source</code> | номер процесса-отправителя |
| <code>tag</code> | идентификатор сообщения |
| <code>comm</code> | коммуникатор области связи |

■ Выходные параметры:

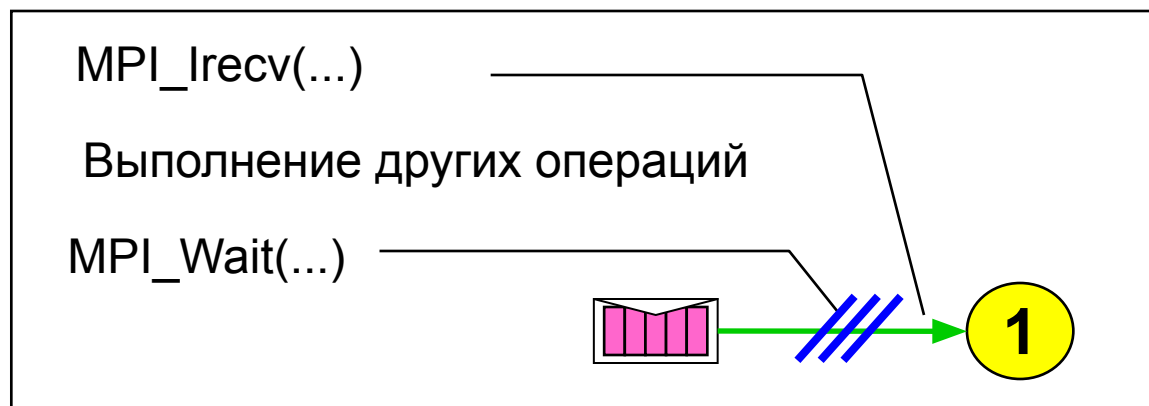
| | |
|----------------------|--|
| <code>rbuf</code> | адрес начала расположения принимаемого сообщения |
| <code>request</code> | запрос обмена |


MPI – Non-Blocking Examples

- неблокирующая посылка



- неблокирующий прием



 = прекращается выполнение операций
пока не завершится передача/прием

Функция ожидания завершения неблокирующей операции **MPI_Wait**

```
int MPI_Wait ( MPI_Request  *request, MPI_Status  
*status)
```

- Входные параметры:

| | |
|----------------------|---------------|
| <code>request</code> | запрос обмена |
|----------------------|---------------|

- Выходные параметры:

| | |
|---------------------|------------------------------|
| <code>status</code> | атрибуты принятого сообщений |
|---------------------|------------------------------|

Функция проверки завершения неблокирующей операции **MPI_Test**

```
int MPI_Test ( MPI_Request *request, int *flag,  
MPI_Status *status)
```

- Входные параметры:

| | |
|----------------------|---------------|
| <code>request</code> | запрос обмена |
|----------------------|---------------|

- Выходные параметры:

| | |
|---------------------|---|
| <code>status</code> | атрибуты принятого сообщений |
| <code>flag</code> | признак завершенности проверяемой операции <code>flag==0</code> , если операция не завершена <code>flag<>0</code> , если операция завершена |

MPI_Sendrecv – совмещение отправки и получения

```
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)
```

■ Выходные параметры:

| | |
|----------------|--|
| <i>recvbuf</i> | адрес начала расположения принимаемого сообщения |
| <i>status</i> | атрибуты принятого сообщения |

- При использовании блокирующего режима передачи сообщений существует потенциальная опасность возникновения тупиковых ситуаций, в которых операции обмена данными блокируют друг друга.
- В ситуациях, когда требуется выполнить *взаимный обмен данными между процессами*, безопаснее использовать *совмещенную операцию MPI_Sendrecv*.

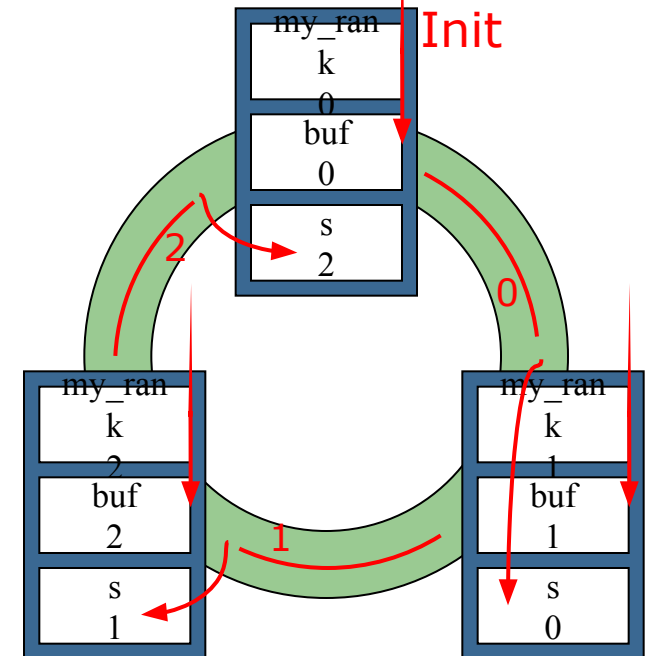
Задание 1

- Нулевой процесс выполняет продолжительный цикл и после его выполнения посылает первому процессору значение вычислений цикла при помощи коммуникационной функции **MPI_Send**;
- А) Первый процесс засекает время t_1 , выполняет блокирующую функцию **MPI_Recv**, засекает время t_2 , выводит присланное значение и затраченное время на ожидание и прием посылки.
- В) Первый процесс засекает время t_1 , выполняет неблокирующую функцию **MPI_Irecv**, засекает время t_2 , выводит полученное значение и затраченное время на выполнение неблокирующей операции приема.
- С) Первый процессор засекает время t_1 , выполняет неблокирующую функцию **MPI_Irecv**, засекает время t_2 , выполняет операцию **MPI_Wait**, засекает время t_3 , выводит полученное значение и затраченное время на выполнение неблокирующей операции приема и ожидание получения посылки.

Задание 2 — Пересылка данных по кольцу

- Каждый процессор помещает свой ранг в целочисленную переменную *buf*.
- Каждый процессор пересылает переменную *buf* соседу справа.
- Каждый процессор суммирует принимаемое значение в переменную *s*, а затем передаёт принятое значение соседу справа.
- Пересылки по кольцу прекращаются, когда каждый процессор получит то значение, с которого начал пересылки:
 - т.е. каждый процессор просуммирует ранги всех процессоров.
- С целью исключения взаимоблокировки используются неблокирующие пересылки **MPI_Isend**.

```
next = (my_id+1) % np;  
prev = (my_id-1+np) % np;
```



Задание 3 — Пересылка данных по кольцу №2

- Замените в предыдущей задаче схему «Isend-Recv-Wait» на Sendrecv.