Передача сообщений.
Операции типа точка-точка.

В операциях типа точка-точка участвуют два процесса, один является отправителем сообщения, другой – получателем. Процесс-отправитель должен вызвать одну из процедур передачи данных и явно указать номер процесса-получателя в некотором коммуникаторе, а процесс-получатель должен вызвать одну из процедур приема с указанием того же коммуникатора. Он может не знать точный номер процесса-отправителя в данном коммуникаторе. Все процедуры делятся на два класса: процедуры с блокировкой и процедуры без блокировки (асинхронные). Процедуры обмена с блокировкой приостанавливают работу процесса до выполнения некоторого условия, а возврат из асинхронных процедур происходит немедленно после инициализации соответствующей коммуникационной операции.

Прием и передача сообщений с блокировкой.

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

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

Прием и передача сообщений с блокировкой.

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

По значению параметра status определяет число count уже принятых (после обращения к MPI_Recv) или принимаемых (после обращения к MPI_Probe или MPI_Iprobe)элементов сообщения типа datatype.

Последовательность сообщений MPI

Если один процесс последовательно посылает два сообщения, соответствующие одному и тому же вызову MPI_Recv, другому процессу, то первым будет принято сообщение, которое было отправлено раньше.

Если два сообщения были одновременно отправлены разными процессами, то порядок их получения принимающим процессом заранее не определён.

Прием и передача сообщений с блокировкой.

Если число реально принятых элементов сообщения меньше count, то в буфере buf изменятся только элементы, соответствующие элементам принятого сообщения. Если количество элементов в принимаемом сообщении больше count, то возникает ошибка переполнения. Блокировка при приеме данных гарантирует, что после возврата из процедуры MPI_Recv все элементы сообщения уже будут приняты и расположены в буфере buf.

Прием и передача сообщений с блокировкой.

Если при приеме сообщения пользователя не интересует заполнение структуры status, то вместо соответствующего аргумента можно указать предопределенную

константу MPI_STATUS_IGNORE. Это также позволит сэкономить немного времени, требуемого на запись соответствующих полей.

Вместо аргументов source и msgtag можно использовать константы:

- MPI_ANY_SOURCE — признак того, что подходит сообщение от любого процесса;
- MPI_ANY_TAG — признак того, что подходит сообщение с любым идентификатором.

Если аргумент статус был передан MPI_Recv, то можно посмотреть параметры приема:

- status.MPI_SOURCE —номер процесса-отправителя
- status.MPI_TAG — идентификатор сообщения
- status.MPI_ERROR — код ошибки

Прием и передача сообщений с блокировкой.

int MPI_Probe(int source, int msgtag, MPI_Comm comm, MPI_Status *status)

Получение в параметре status информации о структуре ожидаемого сообщения с блокировкой. Возврата не произойдёт, пока сообщение с подходящим идентификатором и номером процесса-отправителя не будет доступно для получения.

Получение информации об атрибутах сообщения

```
#include <stdio.h>
#include "mpi.h"
int main(int argc, char **argv)
{
   int rank, size, ibuf;
   MPI_Status status;
   float rbuf;
   MPI_Init(&argc, &argv);
   MPI_Comm_size(MPI_COMM_WORLD, &size);
   MPI_Comm_rank(MPI_COMM_WORLD, &rank);
   ibuf = rank;
   rbuf = 1.0 * rank;
   if(rank==1) MPI_Send(&ibuf, 1, MPI_INT, 0, 5, MPI_COMM_WORLD);
   if(rank==2) MPI_Send(&rbuf, 1, MPI_FLOAT, 0, 5, MPI_COMM_WORLD);
   if(rank==0){
      MPI_Probe(MPI_ANY_SOURCE, 5, MPI_COMM_WORLD, &status);
      if(status.MPI_SOURCE == 1){
         MPI_Recv(&ibuf, 1, MPI_INT, 1, 5, MPI_COMM_WORLD, &status);
         MPI_Recv(&rbuf, 1, MPI_FLOAT, 2, 5, MPI_COMM_WORLD, &status);
      }
      else if(status.MPI_SOURCE == 2){
         MPI_Recv(&rbuf, 1, MPI_FLOAT, 2, 5, MPI_COMM_WORLD, &status);
         MPI_Recv(&ibuf, 1, MPI_INT, 1, 5, MPI_COMM_WORLD, &status);
      }
      printf("Process 0 recv %d from process 1, %f from process 2\n", ibuf, rbuf);
   }
   MPI_Finalize();
}
```

Передача сообщения несуществующему процессу.

Специальное значение MPI_PROC_NULL для несуществующего процесса.

Операции с таким процессом завершаются немедленно с кодом завершения MPI_SUCCESS.

Передача сообщения несуществующему процессу.

```
#include <stdio.h>
#include "mpi.h"
int main(int argc, char **argv)
{
   int size, rank, next, prev, rbuf;
   MPI_Status status;
   MPI_Init(&argc, &argv);
   MPI_Comm_size(MPI_COMM_WORLD, &size);
   MPI_Comm_rank(MPI_COMM_WORLD, &rank);
   next = rank+1;
   if(next == size) next = MPI_PROC_NULL;
   MPI_Send(&rank, 1, MPI_INT, next, 5, MPI_COMM_WORLD);
   prev = rank-1;
   if(prev == -1) prev = MPI_PROC_NULL;
   rbuf = -1;
   MPI_Recv(&rbuf, 1, MPI_INT, prev, 5, MPI_COMM_WORLD, &status);
   printf("process %d rbuf = %d\n", rank, rbuf);
   MPI_Finalize();
}
```

Тупиковые ситуации

Самая простая тупиковая ситуация (deadlock) может возникнуть, если в двух процессах будет последовательно вызван Recv а потом Send.

Процесс 0:	Процесс 1:
Recv (1)	Recv (0)
Send (1)	Send (0)

Тупиковые ситуации

Может показаться, что перестановка Send и Recv исправит ситуацию.

Процесс 0:
Send (1)
Recv (1)

Процесс 1:
Send (0)
Recv (0)

Тупиковые ситуации

Для разрешения подобной тупиковой ситуации следует использовать следующую схему:

Процесс 0:	Процесс 1:
Send (1)	Recv (0)
Recv (1)	Send (0)

Модификации процедуры MPI_Send

MPI_Bsend — передача сообщения с буферизацией.

MPI_Ssend — передачасообщенияс синхронизацией.

MPI_Rsend — передача сообщения по готовности.

Модификации процедуры MPI_Send - MPI_Bsend

MPI_Bsend —передача сообщения с буферизацией. Если прием сообщения еще не был инициализирован, то сообщение будет записано в специальный буфер, и произойдет немедленный возврат. Процедура может вернуть код ошибки, если места под буфер недостаточно. О выделении массива для буферизации должен заботиться пользователь.

Модификации процедуры MPI_Send - MPI_Bsend

Добавление буфера осуществляется процедурой int MPI_Buffer_attach(void* buf, int size).

Размер массива, выделяемого для буферизации, должен превосходить общий размер сообщения как минимум на величину, определяемую константой MPI_BSEND_OVERHEAD.

Модификации процедуры MPI_Send - MPI_Bsend

Для освобождения памяти, выделенной под буфер используется процедура
int MPI_Buffer_detach(void* buf, int* size).
Процедура возвращает в аргументах начало освобожденного массива buf и size размер этого массива. Вызов процесса блокируется до того момента, когда все сообщения уйдут из данного буфера. MPI_Buffer_attach и MPI_Buffer_detach для аргумента buf используют один и тот же тип void* для того чтобы избежать сложностей приведения типов.

Модификации процедуры MPI_Send - MPI_Bsend

MPI_Buffer_attach и MPI_Buffer_detach для аргумента buf используют один и тот же тип void* для того чтобы избежать сложностей приведения типов.

```
#define BUFFSIZE 10000
int size
char *buff;
MPI_Buffer_attach(malloc(BUFFSIZE), BUFFSIZE);
/* буфер на 10000 байтов может теперь быть  использован MPI_Bsend */
MPI_Buffer_detach(&buff, &size);
/* размер буфера уменьшен до нуля */
MPI_Buffer_attach(buff, size);
/* буфер на  10000 байтов доступен снова */
```

В примере `&buff,` который имеет тип **char****, может быть передан как аргумент `MPI_Buffer_detach` без приведения типов. Если бы формальный параметр этой процедуры имел тип `void**`, нужно было бы выполнить приведение типов перед и после вызова.

Пример механизма работы функций MPI_Buffer_attach и MPI_Buffer_detach.

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

char * buffer;

void f1( void * buff){
      buffer = (char*)buff;
}

void f2( void * buff){
      *((char**)buff) = buffer;
}

int main(void){

      void* b = malloc (10 * sizeof(char));

      printf("%p \n", b);
      f1(b);

      b = NULL;
      printf("%p \n", b);

      f2(&b);
      printf("%p \n", b);

       free(b);

      return 0;
}
```

Пример MPI_Bsend

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

int main(int argc, char **argv) {
int BUFSIZE = sizeof(int) + MPI_BSEND_OVERHEAD; char *buf;
int rank, ibufsize, rbuf, size;
MPI_Status status;
MPI_Init(&argc, &argv);
MPI_Comm_size(MPI_COMM_WORLD, &size);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
   if(rank == 0){
       MPI_Buffer_attach(malloc(BUFSIZE), BUFSIZE);
       MPI_Bsend(&rank, 1, MPI_INT, 1, 5, MPI_COMM_WORLD);
       MPI_Buffer_detach(&buf, &ibufsize);
       free(buf);
   }
   if(rank == 1){
      MPI_Recv(&rbuf, 1, MPI_INT, 0, 5, MPI_COMM_WORLD, &status);
      printf("Process 1 received %d from process %d\n",rbuf, status.MPI_SOURCE);
    }
    MPI_Finalize();
}
```

Модификации процедуры MPI_Send - MPI_Ssend

MPI_Ssend —передача сообщения с синхронизацией.

Выход из процедуры произойдет только тогда, когда прием сообщения будет инициализирован процессом-получателем. Использование передачи с синхронизацией может замедлить выполнение программы, но позволяет избежать наличия в системе большого количества не принятых буферизованных сообщений.

Модификации процедуры MPI_Send - MPI_Rsend

MPI_Rsend —передача сообщения по готовности.

Данной процедурой можно пользоваться только в том случае, если процесс-получатель уже инициировал прием сообщения. Иначе вызов процедуры является ошибочным и результат её выполнения не детерминирован.

Модификации процедуры MPI_Send - MPI_Rsend

Гарантировать инициализацию приема сообщения перед вызовом процедуры MPI_Rsend можно с помощью операций, осуществляющих явную или неявную синхронизацию процессов (например, MPI_Barrier или MPI_Ssend). Во многих реализациях процедура MPI_Rsend сокращает протокол взаимодействия между отправителем и получателем, уменьшая накладные расходы на передачу данных.