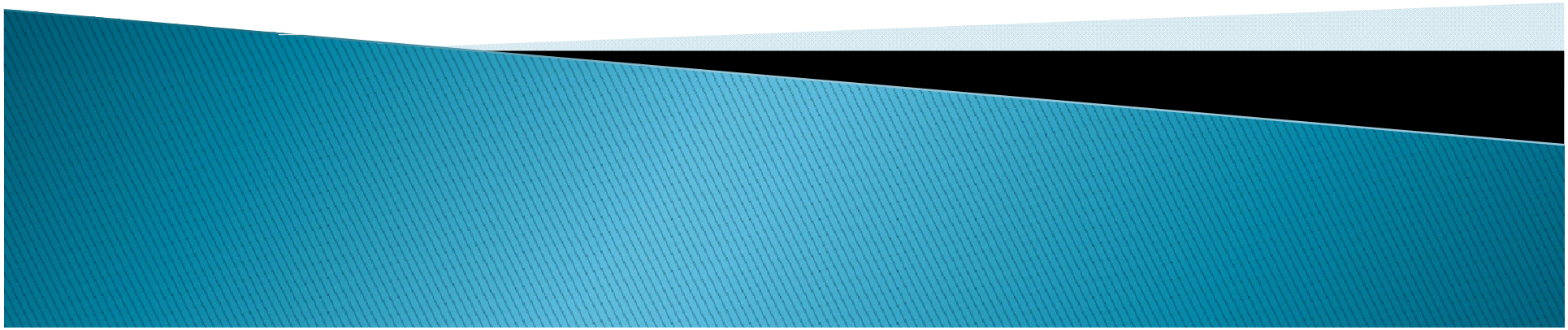
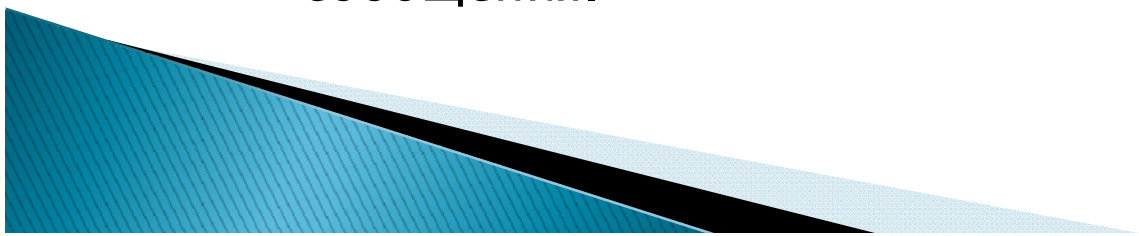
Обзор технологии параллельного программирования MPI



Двухточечный (point-to-point, p2p) обмен

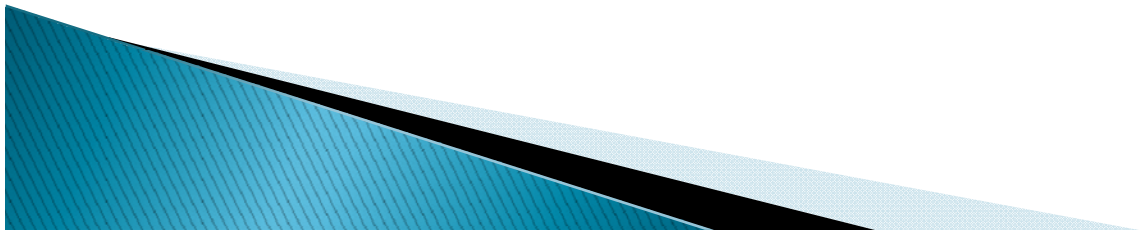
- Правильно организованный двухточечный обмен сообщениями должен исключать возможность блокировки или некорректной работы параллельной MPI-программы.

- Примеры ошибок в организации двухточечных обменов:
  - выполняется передача сообщения, но не выполняется его прием;
  - процесс-источник и процесс-получатель одновременно пытаются выполнить блокирующие передачу или прием сообщения.



Двухточечный (point-to-point, p2p) обмен

```
#include <mpi.h>
int main (int argc, char **argv)
{
    int rank, size;
    int recv, send, left, right;
    MPI_Status status;
    MPI_Init( &argc, &argv );
    MPI_Comm_size( MPI_COMM_WORLD, &size );
    MPI_Comm_rank( MPI_COMM_WORLD, &rank );
    left = (rank > 0)? rank - 1 : size-1;
    right = (rank + 1)%size ;
    MPI_Recv( &recv, 1, MPI_INT, left, 100, MPI_COMM_WORLD, &status );
    send = rank;
    MPI_Send( &send, 1, MPI_INT, right, 100, MPI_COMM_WORLD );
    MPI_Finalize( );
    return 0;
}
```



Двухточечный (point-to-point, p2p) обмен

В MPI приняты следующие соглашения об именах подпрограмм двухточечного обмена:

MPI_[I][R, S, B]Send

здесь префикс [I] (Immediate) обозначает неблокирующий режим.

Один из префиксов [R, S, B] обозначает режим обмена: по готовности, синхронный и буферизованный.

Отсутствие префикса обозначает подпрограмму стандартного обмена.

Имеется 8 разновидностей операции передачи сообщений.

Для подпрограмм приема:

MPI_[I]Recv

то есть всего 2 разновидности приема.

Подпрограмма MPI_Irsend, например, выполняет передачу «по готовности» в неблокирующем режиме, MPI_Bsend буферизованную передачу с блокировкой, а MPI_Recv выполняет блокирующий прием сообщений.

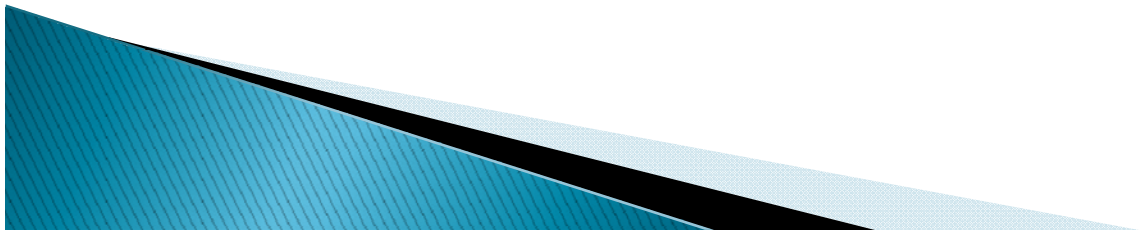
Подпрограмма приема любого типа может принять сообщения от любой подпрограммы передачи.

Стандартная блокирующая передача

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

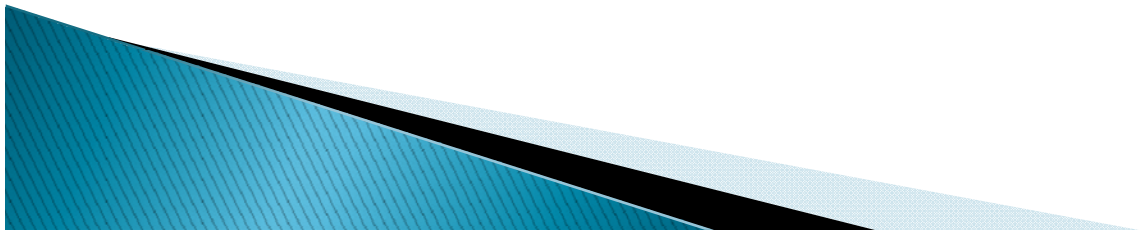
MPI_Send(buf, count, datatype, dest, tag, comm,ierr)

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



Стандартная блокирующая передача

- При стандартной блокирующей передаче после завершения вызова (после возврата из функции/процедуры передачи) можно использовать любые переменные, использовавшиеся в списке параметров. Такое использование не повлияет на корректность обмена.
- Дальнейшая «судьба» сообщения зависит от реализации MPI. Сообщение может быть сразу передано процессу-получателю или может быть скопировано в буфер передачи.
- Завершение вызова не гарантирует доставки сообщения по назначению.



Стандартный блокирующий прием

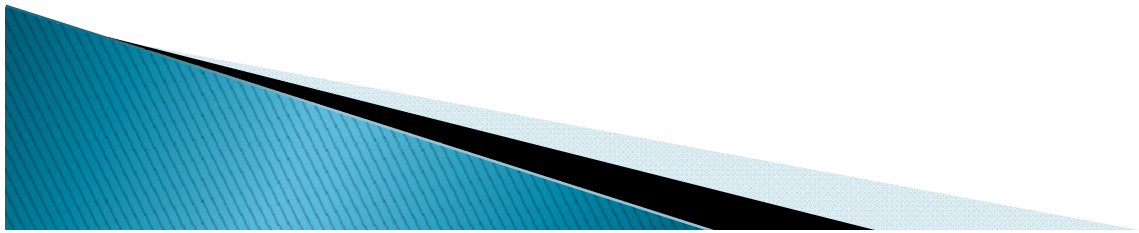
int MPI_Recv(void *buf, int count, MPI_Datatype datatype, int source, int tag, MPI_Comm comm, MPI_Status *status)

MPI_Recv(buf, count, datatype, dest, tag, comm, status, ierr)

- buf – адрес первого элемента в буфере приёма;
- count – количество элементов в буфере приёма;
- datatype – тип MPI каждого пересылаемого элемента;
- source – ранг процесса-отправителя сообщения (целое число от 0 до n – 1, где n число процессов в области взаимодействия);
- tag – тег сообщения;
- comm – коммуникатор;
- status – статус обмена;
- ierr – код завершения.

Стандартный блокирующий прием

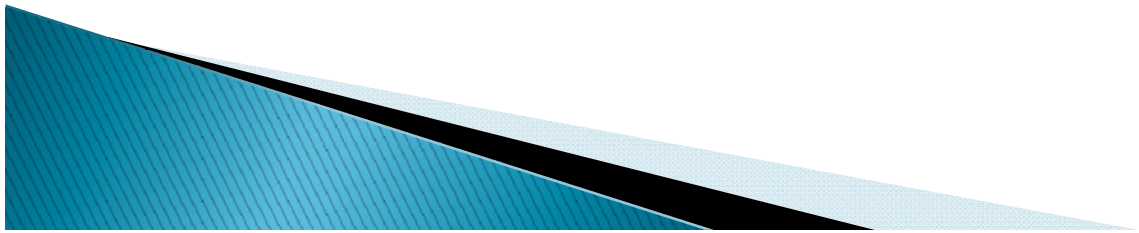
- Значение параметра count может оказаться больше, чем количество элементов в принятом сообщении. В этом случае после выполнения приёма в буфере изменится значение только тех элементов, которые соответствуют элементам фактически принятого сообщения.
- Для функции MPI_Recv гарантируется, что после завершения вызова сообщение принято и размещено в буфере приема.



Прием сообщения

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.



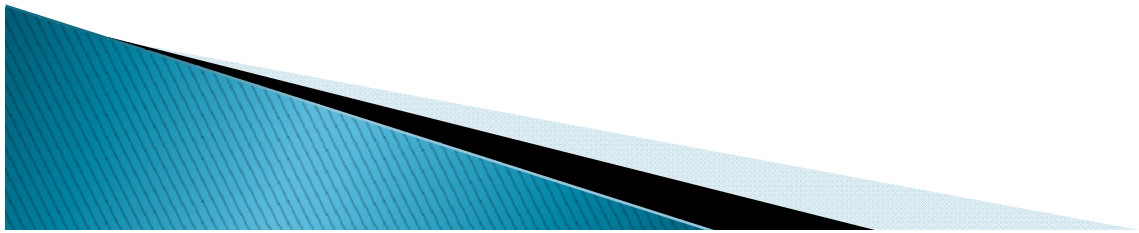
Коды завершения

- MPI_ERR_COMM – неправильно указан коммуникатор. Часто возникает при использовании «пустого» коммуникатора;
- MPI_ERR_COUNT – неправильное значение аргумента count (количество пересылаемых значений);
- MPI_ERR_TYPE – неправильное значение аргумента, задающего тип данных;
- MPI_ERR_TAG – неправильно указан тег сообщения;
- MPI_ERR_RANK – неправильно указан ранг источника или адресата сообщения;
- MPI_ERR_ARG – неправильный аргумент, ошибочное задание которого не попадает ни в один класс ошибок;
- MPI_ERR_REQUEST – неправильный запрос на выполнение операции.

Джокеры

- В качестве ранга источника сообщения и в качестве тега сообщения можно использовать «джокеры» :
  - MPI_ANY_SOURCE – любой источник;
  - MPI_ANY_TAG – любой тег.

- При использовании «джокеров» есть опасность приема сообщения, не предназначенного данному процессу

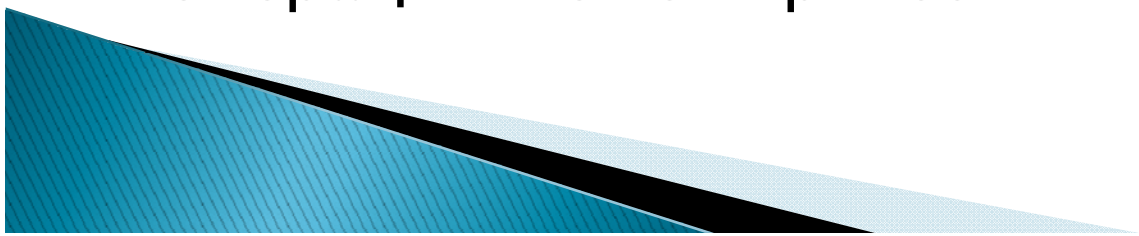


Двухточечные обмены

Подпрограмма MPI_Recv может принимать сообщения, отправленные в любом режиме.

Прием может выполняться от произвольного процесса, а в операции передачи должен быть указан вполне определенный адрес.

Приемник может использовать «джокеры» для источника и для тега. Процесс может отправить сообщение и самому себе, но следует учитывать, что использование в этом случае блокирующих операций может привести к «тупику».



Двухточечные обмены

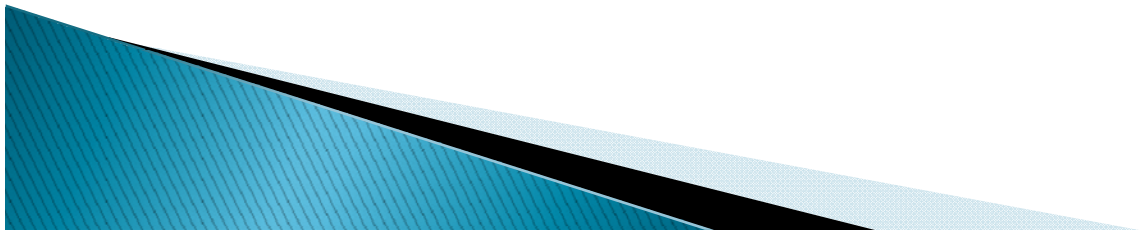
Размер полученного сообщения (count) можно определить с помощью вызова подпрограммы

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

MPI_Get_count(status, datatype, count, ierr)

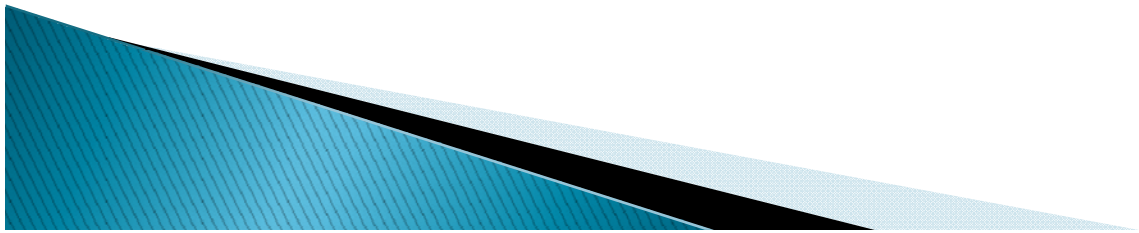
- count – количество элементов в буфере передачи;
- datatype – тип MPI каждого пересылаемого элемента;
- status – статус обмена;
- ierr – код завершения.

Аргумент datatype должен соответствовать типу данных, указанному в операции обмена



Двухточечный обмен с буферизацией

- Передача сообщения в буферизованном режиме может быть начата независимо от того, зарегистрирован ли соответствующий прием. Источник копирует сообщение в буфер, а затем передает его в неблокирующем режиме, так же как в стандартном режиме.
- Эта операция локальна, поскольку ее выполнение не зависит от наличия соответствующего приема.
- Если объем буфера недостаточен, возникает ошибка. Выделение буфера и его размер контролируются программистом.



Двухточечный обмен с буферизацией

- Размер буфера должен превосходить размер сообщения на величину MPI_BSEND_OVERHEAD. Это дополнительное пространство используется подпрограммой буферизованной передачи для своих целей.

- Если перед выполнением операции буферизованного обмена не выделен буфер, MPI ведет себя так, как если бы с процессом был связан буфер нулевого размера. Работа с таким буфером обычно завершается сбоем программы.

- Буферизованный обмен рекомендуется использовать в тех ситуациях, когда программисту требуется больший контроль над распределением памяти. Этот режим удобен и для отладки, поскольку причину переполнения буфера определить легче, чем причину тупика.

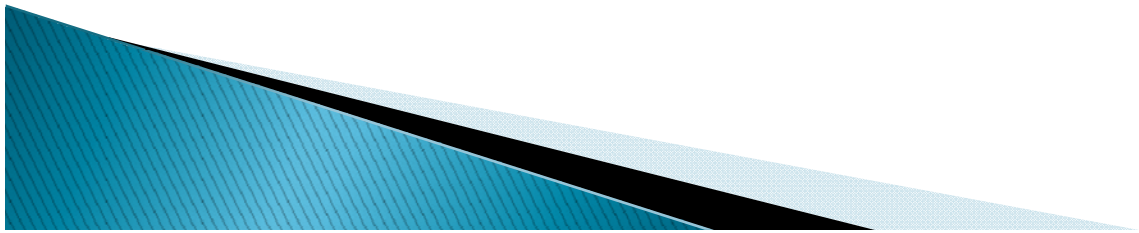
Двухточечный обмен с буферизацией

- При выполнении буферизованного обмена программист должен заранее создать буфер достаточного размера:

```
int MPI_Buffer_attach(void *buf, size)
```

```
MPI_Buffer_attach(buf, size, ierr)
```

- В результате вызова создается буфер `buf` размером `size` байтов. В программах на языке Fortran роль буфера может играть массив. За один раз к процессу может быть подключен только один буфер.

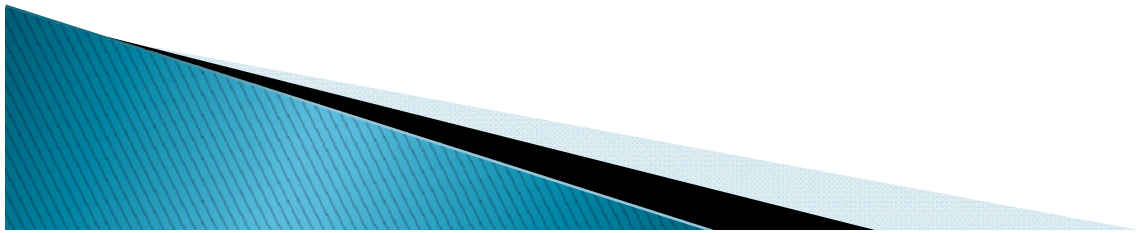


Двухточечный обмен с буферизацией

- Буферизованная передача завершается сразу, поскольку сообщение немедленно копируется в буфер для последующей передачи:

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

MPI_Bsend(buf, count, datatype, dest, tag,
  comm, ierr)
```



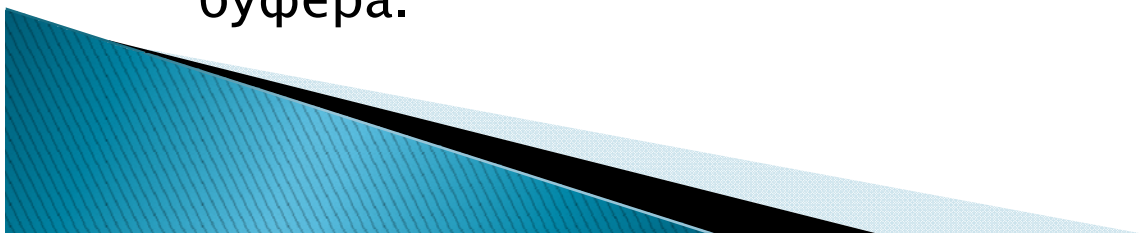
Двухточечный обмен с буферизацией

- После завершения работы с буфером его необходимо отключить:

```
int MPI_Buffer_detach(void *buf, int *size)

MPI_Buffer_detach(buf, size, ierr)
```

- Возвращается адрес (`buf`) и размер отключаемого буфера (`size`). Эта операция блокирует работу процесса до тех пор, пока все сообщения, находящиеся в буфере, не будут обработаны. Вызов данной подпрограммы можно использовать для форсированной передачи сообщений. После завершения вызова можно вновь использовать память, которую занимал буфер. В языке С данный вызов не освобождает автоматически память, отведенную для буфера.



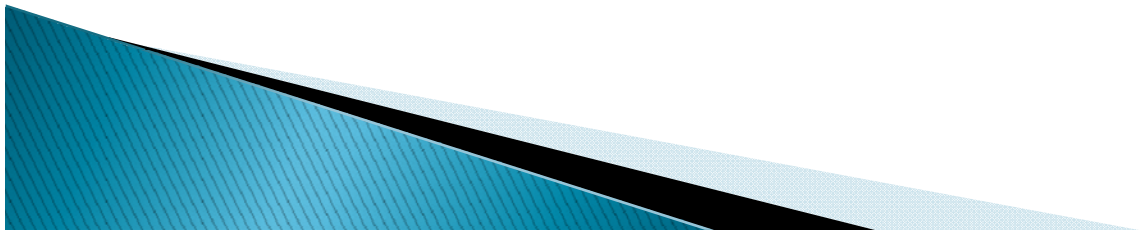
Двухточечный обмен с буферизацией

```
#include <mpi.h>
#include <stdio.h>
#define M 10
int main( int argc, char **argv )
{
    int n;
    int rank, 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 ) {
       int blen = M * (sizeof(int) + MPI_BSEND_OVERHEAD);
       int *buf = (int*) malloc(blen);
       MPI_Buffer_attach (buf, blen);
       for(int i = 0; i < M; i ++) {
            n = i;
            MPI_Bsend (&n, 1, MPI_INT, 1, i, MPI_COMM_WORLD );
        }
        MPI_Buffer_detach(&buf, &blen);
        free(buf);
    }
    else if ( rank == 1) {
        for(int i = 0; i < M; i ++) {
            MPI_Recv (&n, 1, MPI_INT, 0, i,
                MPI_COMM_WORLD,&status );
        }
    }
    MPI_Finalize();
    return 0;
}
```

Синхронный режим

- Завершение передачи происходит только после того, как прием сообщения инициализирован другим процессом.
- Посылающая сторона запрашивает у принимающей стороны подтверждение выдачи операции receive – «квитанцию».

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



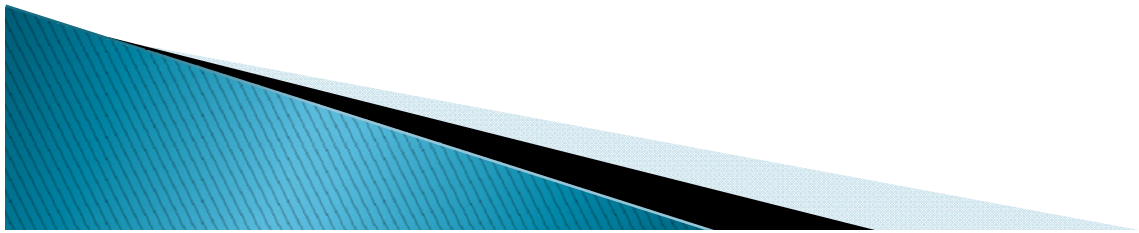
Режим «по готовности»

Передача «по готовности» выполняется с помощью подпрограммы

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

```
MPI_Rsend(buf, count, datatype, dest, tag, comm, ierr)
```

Передача «по готовности» должна начинаться, если уже зарегистрирован соответствующий прием. При несоблюдении этого условия результат выполнения операции не определен.

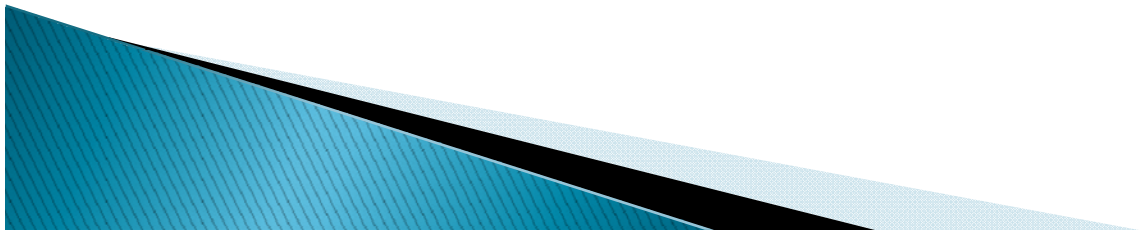


Режим «по готовности»

Завершается она сразу же. Если прием не зарегистрирован, результат выполнения операции не определен.

Завершение передачи не зависит от того, вызвана ли другим процессом подпрограмма приема данного сообщения или нет, оно означает только, что буфер передачи можно использовать вновь.

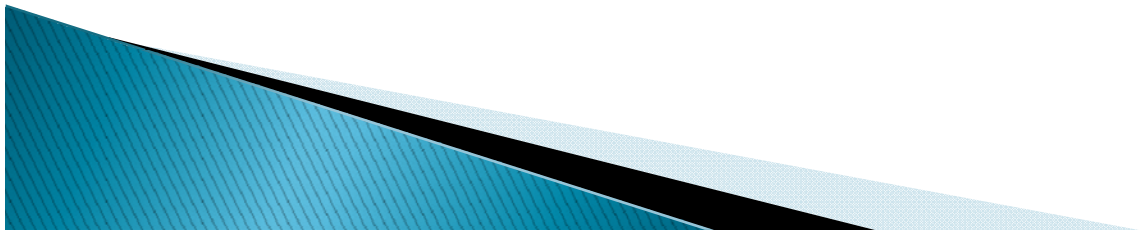
Сообщение просто выбрасывается в коммуникационную сеть в надежде, что адресат его получит. Эта надежда может и не сбыться.



Режим «по готовности»

Обмен «по готовности» может увеличить производительность программы, поскольку здесь не используются этапы установки межпроцессных связей, а также буферизация.

Все это — операции, требующие времени. С другой стороны, обмен «по готовности» потенциально опасен, кроме того, он усложняет отладку, поэтому его рекомендуется использовать только в том случае, когда правильная работа программы гарантируется ее логической структурой, а выигрыша в быстродействии надо добиться любой ценой.



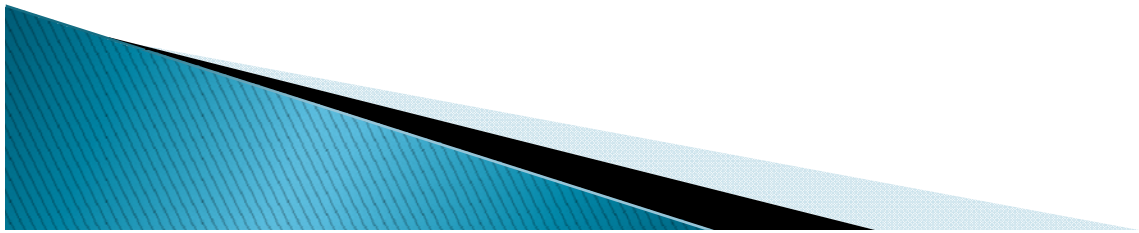
Совместные прием и передача

- Подпрограмма MPI_Sendrecv выполняет прием и передачу 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)
```

- Подпрограмма MPI_Sendrecv_replace выполняет прием и передачу данных, используя общий буфер для передачи и приёма:

```
int MPI_Sendrecv_replace(void *buf, int count,
MPI_Datatype datatype, int dest, int sendtag, int
source,int recvtag, MPI_Comm comm, MPI_Status
*status)
```



Неблокирующие обмены

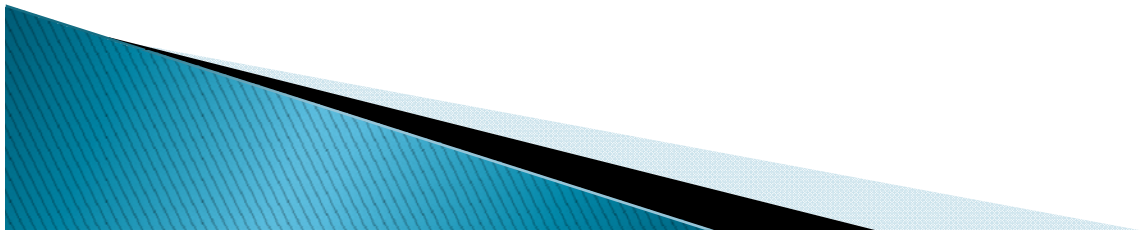
- Вызов подпрограммы неблокирующей передачи инициирует, но не завершает ее. Завершиться выполнение подпрограммы может еще до того, как сообщение будет скопировано в буфер передачи.

- Применение неблокирующих операций улучшает производительность программы, поскольку в этом случае допускается перекрытие (то есть одновременное выполнение) вычислений и обменов. Передача данных из буфера или их считывание может происходить одновременно с выполнением процессом другой работы.

Неблокирующие обмены

- Для завершения неблокирующего обмена требуется вызов дополнительной процедуры, которая проверяет, скопированы ли данные в буфер передачи.

- **ВНИМАНИЕ!**
  При неблокирующем обмене возвращение из подпрограммы обмена происходит сразу, но запись в буфер или считывание из него после этого производить **нельзя** – сообщение может быть еще не отправлено или не получено и работа с буфером может «испортить» его содержимое.



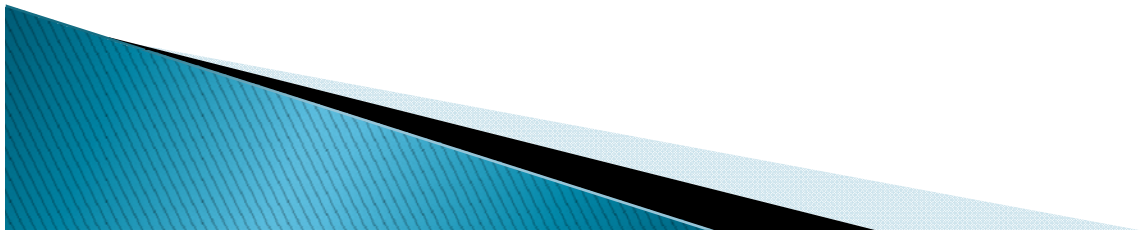
Неблокирующие обмены

Неблокирующий обмен выполняется в два этапа:

1. **инициализация** обмена;
2. **проверка завершения** обмена.

Разделение этих шагов делает необходимым *маркировку* каждой операции обмена, которая позволяет целенаправленно выполнять проверки завершения соответствующих операций.

Для маркировки в неблокирующих операциях используются *идентификаторы операций обмена*



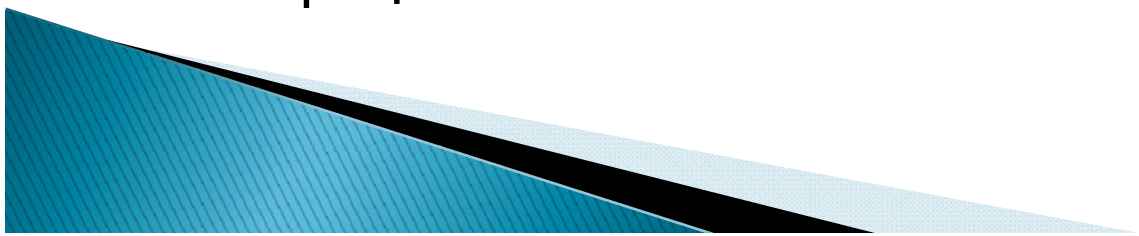
Неблокирующие обмены

- Инициализация неблокирующей стандартной передачи выполняется подпрограммами `MPI_I[S, B, R]send`.
  Стандартная неблокирующая передача выполняется подпрограммой:

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

```
MPI_Isend(buf, count, datatype, dest, tag, comm,
request, ierr)
```

- Входные параметры этой подпрограммы аналогичны аргументам подпрограммы `MPI_Send`.
- Выходной параметр `request` – идентификатор операции.



Неблокирующие обмены

- Инициализация неблокирующего приема выполняется при вызове подпрограммы:

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

```
MPI_Irecv(buf, count, datatype, source, tag,
comm, request, ierr)
```

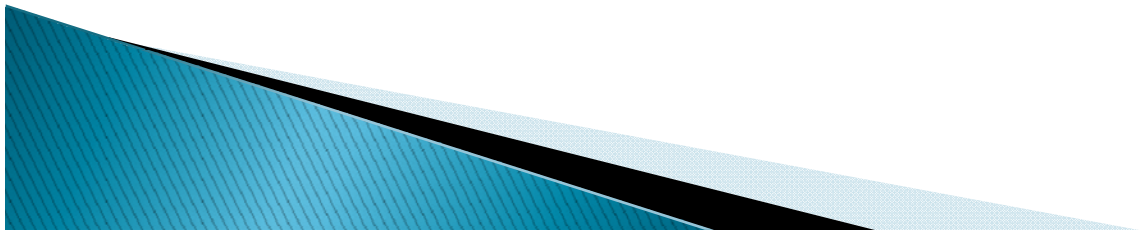
- Назначение аргументов здесь такое же, как и в ранее рассмотренных подпрограммах, за исключением того, что указывается ранг не адресата, а источника сообщения (source).

Неблокирующие обмены

- Вызовы подпрограмм неблокирующего обмена формируют *запрос* на выполнение операции обмена и связывают его с идентификатором операции `request`.

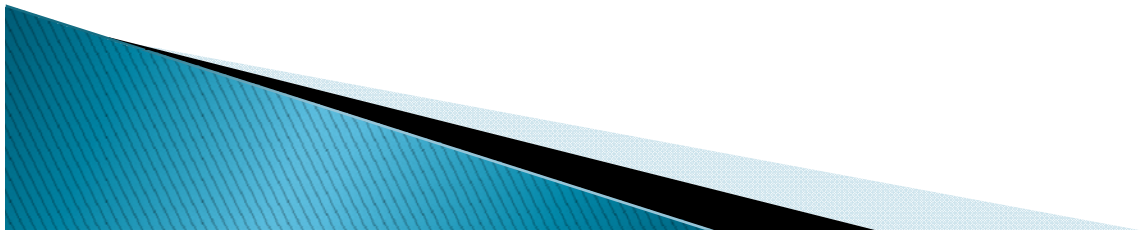
- Запрос идентифицирует свойства операции обмена:
  - режим;
  - характеристики буфера обмена;
  - контекст;
  - тег и ранг.

- Запрос содержит информацию о состоянии ожидающих обработки операций обмена и может быть использован для получения информации о состоянии обмена или для ожидания его завершения.



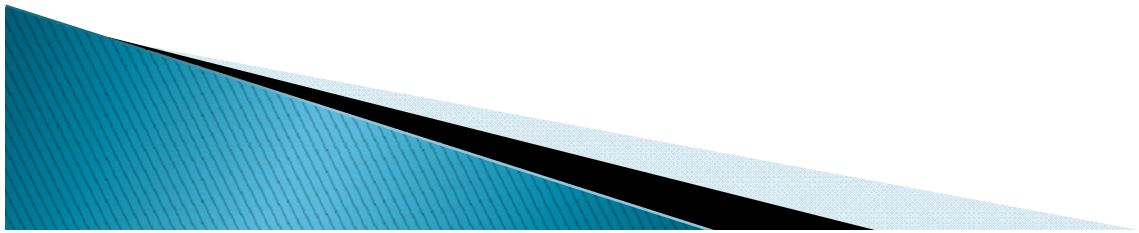
Проверка выполнения обмена

- Проверка фактического выполнения передачи или приема в неблокирующем режиме осуществляется с помощью вызова *подпрограмм ожидания*, блокирующих работу процесса до завершения операции или неблокирующих подпрограмм проверки, возвращающих логическое значение «истина», если операция выполнена



Проверка выполнения обмена

- В том случае, когда одновременно несколько процессов обмениваются сообщениями, можно использовать проверки, которые применяются одновременно к нескольким обменам.
- Есть три типа таких проверок:
  1. проверка завершения всех обменов;
  2. проверка завершения любого обмена из нескольких;
  3. проверка завершения заданного обмена из нескольких.
- Каждая из этих проверок имеет две разновидности:
  1. «ожидание»;
  2. «проверка».

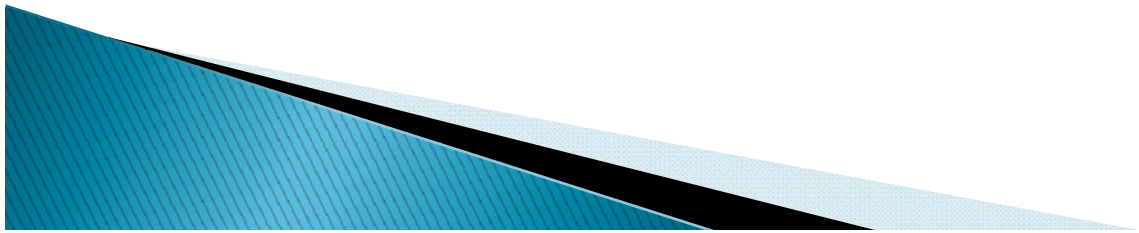


Блокирующие операции проверки

- Подпрограмма MPI_Wait блокирует работу процесса до завершения приема или передачи сообщения:

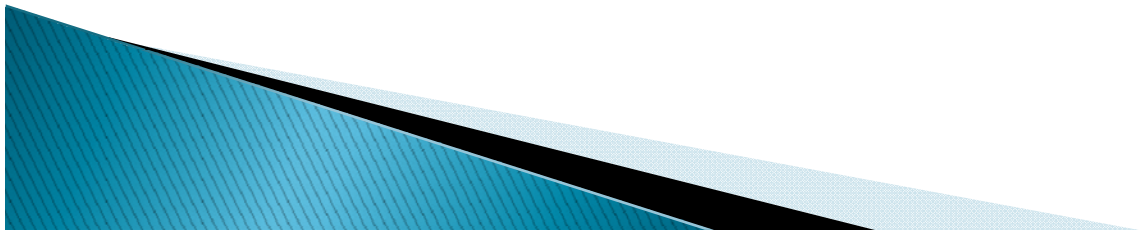
```
int MPI_Wait(MPI_Request *request, MPI_Status
*status)
MPI_Wait(request, status, ierr)
```

- Входной параметр request — идентификатор операции обмена, выходной — статус (status).



Блокирующие операции проверки

- Успешное выполнение подпрограммы `MPI_Wait` после вызова `MPI_Ibsend` подразумевает, что буфер передачи можно использовать вновь, то есть пересылаемые данные отправлены или скопированы в буфер, выделенный при вызове подпрограммы `MPI_Buffer_attach`.
- В этот момент уже нельзя отменить передачу. Если не будет зарегистрирован соответствующий прием, буфер нельзя будет освободить. В этом случае можно применить подпрограмму `MPI_Cancel`, которая освобождает память, выделенную подсистеме коммуникаций.



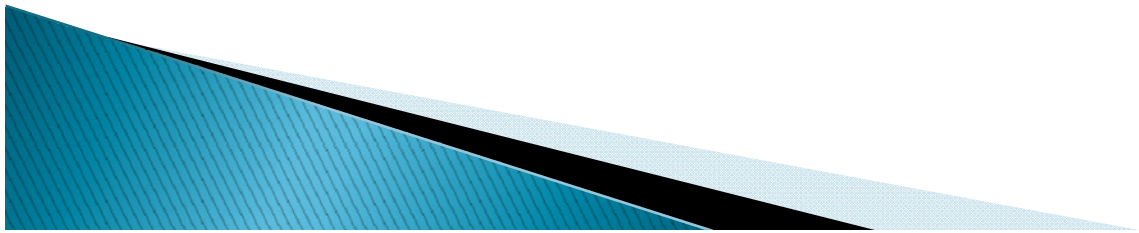
Проверка завершения всех обменов

- Проверка завершения всех обменов выполняется подпрограммой:

```
int MPI_Waitall(int count, MPI_Request
requests[], MPI_Status statuses[])

MPI_Waitall(count, requests, statuses, ierr)
```

- При вызове этой подпрограммы выполнение процесса блокируется до тех пор, пока **все** операции обмена, связанные с активными запросами в массиве `requests`, не будут выполнены. Возвращается статус этих операций. Статус обменов содержится в массиве `statuses`. `count` – количество запросов на обмен (размер массивов `requests` и `statuses`).

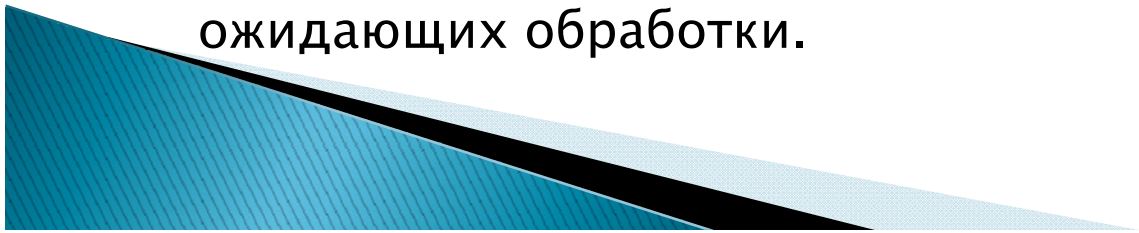


Проверка завершения всех обменов

- В результате выполнения подпрограммы `MPI_Waitall` запросы, сформированные неблокирующими операциями обмена, аннулируются, а соответствующим элементам массива присваивается значение `MPI_REQUEST_NULL`.

- В случае неуспешного выполнения одной или более операций обмена подпрограмма `MPI_Waitall` возвращает код ошибки `MPI_ERR_IN_STATUS` и присваивает полю ошибки статуса значение кода ошибки соответствующей операции.

- Если операция выполнена успешно, полю присваивается значение `MPI_SUCCESS`, а если не выполнена, но и не было ошибки – значение `MPI_ERR_PENDING`. Это соответствует наличию запросов на выполнение операции обмена, ожидающих обработки.

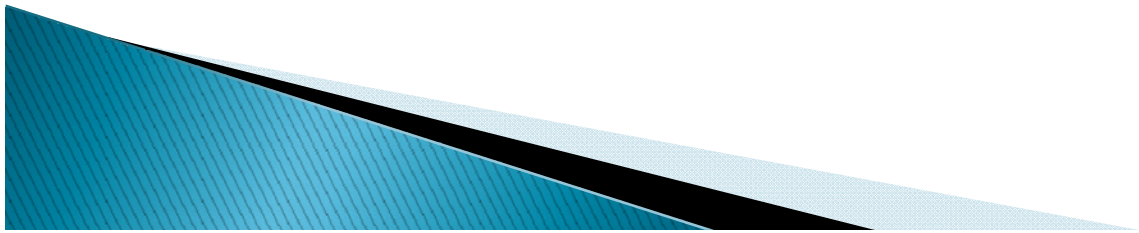


Алгоритм Якоби. Последовательная версия

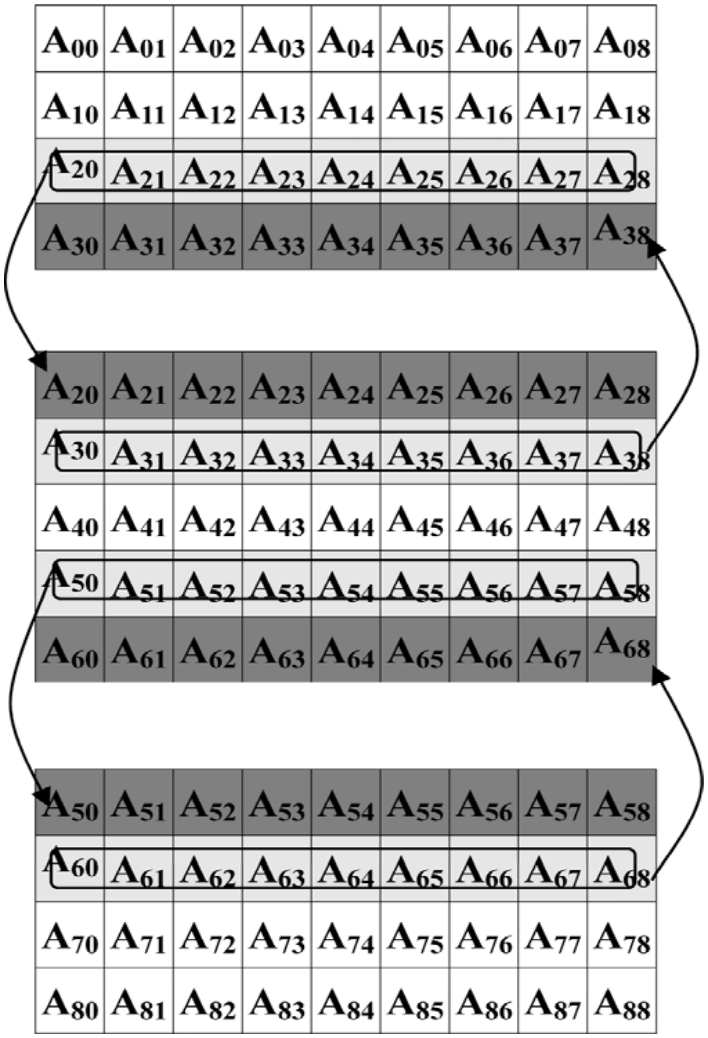
```
/* Jacobi program */
#include <stdio.h>
#define L 1000
#define ITMAX 100
int i,j,it;
double A[L][L];
double B[L][L];
int main(int an, char **as)
{
        printf("JAC STARTED\n");
        for(i=0;i<=L-1;i++)
                for(j=0;j<=L-1;j++)
                {
                        A[i][j]=0.;
                        B[i][j]=1.+i+j;
                }
```

Алгоритм Якоби. Последовательная версия

```
/******  iteration loop  ************************/
for(it=1; it<ITMAX;it++)
{
      for(i=1;i<=L-2;i++)
            for(j=1;j<=L-2;j++)
                  A[i][j] = B[i][j];
      for(i=1;i<=L-2;i++)
            for(j=1;j<=L-2;j++)
                  B[i][j] = (A[i-1][j]+A[i+1][j]+A[i][j-1]+A[i][j+1])/4.;
}
return 0;
}
```



Алгоритм Якоби. MPI-версия



Shadow edges

Imported elememts