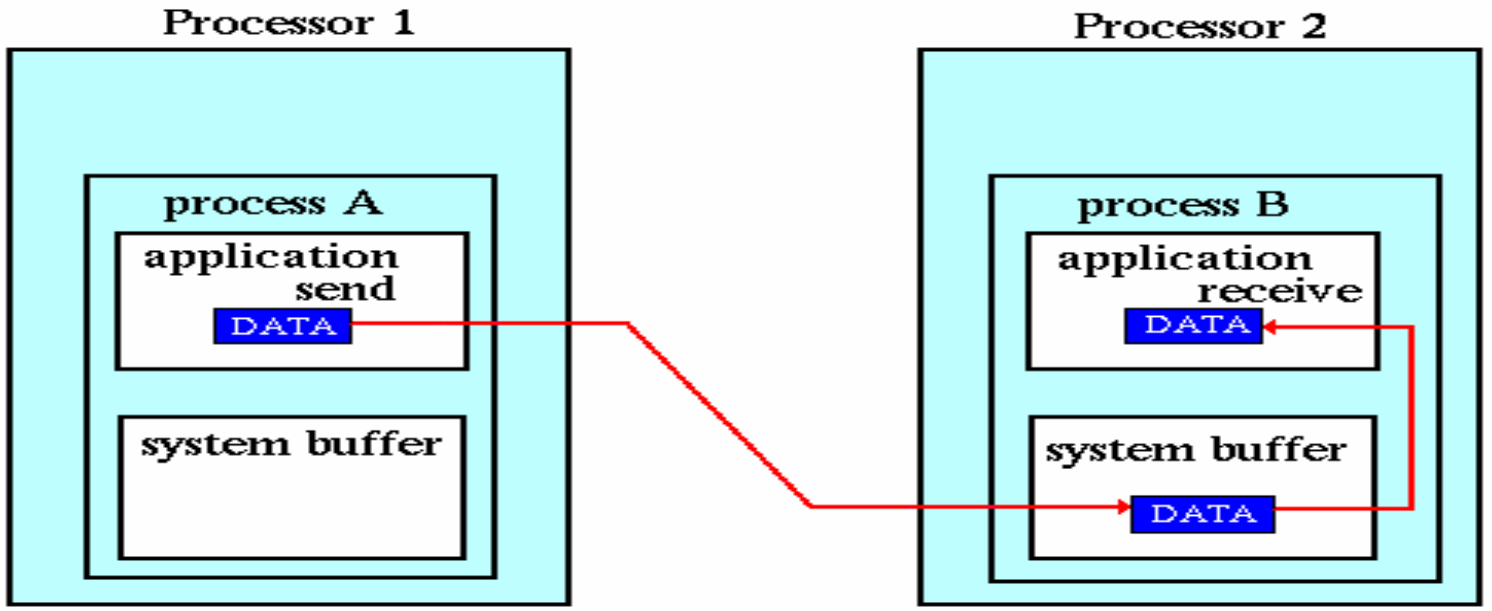
Суперкомпьютерные вычислительные технологии.

Лекционно-практический курс
для студентов 5 курса факультета ВМиК МГУ
сентябрь – декабрь 2013 г.

Лекция 5

4 октября 2013 г.

Схема передачи



Path of a message buffered at the receiving process

Функции управления окружением (Environment Management Routines)

Environment Management Routines		
MPI_Abort	MPI_Errhandler_create	MPI_Errhandler_free
MPI_Errhandler_get	MPI_Errhandler_set	MPI_Error_class
MPI_Error_string	MPI_Finalize	MPI_Get_processor_name
MPI_Init	MPI_Initialized	MPI_Wtick
MPI_Wtime		

Взаимодействие «точка-точка»

- Самая простая форма обмена сообщением
- Один процесс посылает сообщения другому
- Несколько вариантов реализации того, как пересылка и выполнение программы совмещаются

Варианты передачи «точка-точка»

- Синхронные пересылки
- Асинронные передачи
- Блокирующие передачи
- Неблокирующие передачи

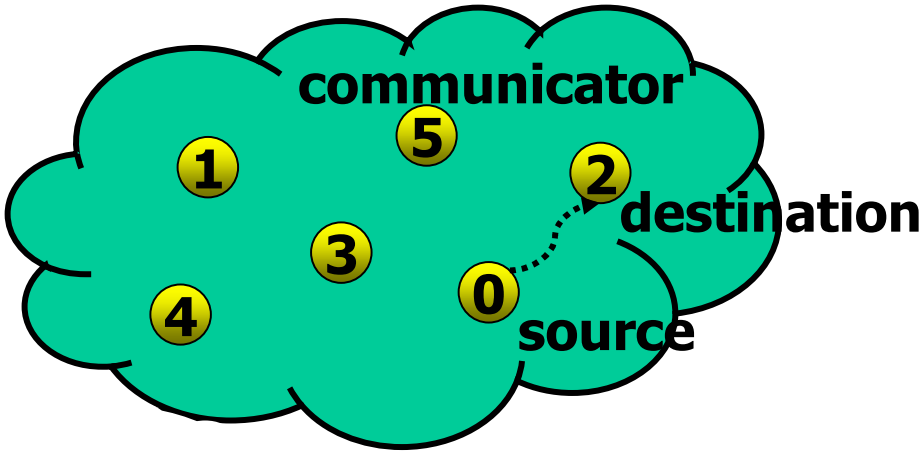
Функции MPI передачи «точка-точка»

Point-to-Point Communication Routines		
MPI_Bsend	MPI_Bsend_init	MPI_Buffer_attach
MPI_Buffer_detach	MPI_Cancel	MPI_Get_count
MPI_Get_elements	MPI_Ibsend	MPI_Iprobe
MPI_Irecv	MPI_Irsend	MPI_Isend
MPI_Issend	MPI_Probe	MPI_Recv
MPI_Recv_init	MPI_Request_free	MPI_Rsend
MPI_Rsend_init	MPI_Send	MPI_Send_init
MPI_Sendrecv	MPI_Sendrecv_replace	MPI_Ssend
MPI_Ssend_init	MPI_Start	MPI_Startall
MPI_Test	MPI_Test_cancelled	MPI_Testall
MPI_Testany	MPI_Testsome	MPI_Wait
MPI_Waitall	MPI_Waitany	MPI_Waitsome

Функции коллективных передач

Collective Communication Routines		
MPI_Allgather	MPI_Allgatherv	MPI_Allreduce
MPI_Alltoall	MPI_Alltoallv	MPI_Barrier
MPI_Bcast	MPI_Gather	MPI_Gatherv
MPI_Op_create	MPI_Op_free	MPI_Reduce
MPI_Reduce_scatter	MPI_Scan	MPI_Scatter
MPI_Scatterv		

Передача сообщений типа «точка-точка»



- Взаимодействие между двумя процессами
- Процесс-отправитель(Source process) ***посылает*** сообщение процессу-получателю (Destination process)
- Процесс-получатель ***принимает*** сообщение
- Передача сообщения происходит в рамках заданного коммуникатора
- Процесс-получатель определяется рангом в коммуникаторе

Блокирующие и неблокирующие передачи

- **Блокирующие**: возврат из функций передачи сообщений только по завершению коммуникаций
- **Неблокирующие** (асинхронные): немедленный возврат из функций, пользователь должен контролировать завершение передач

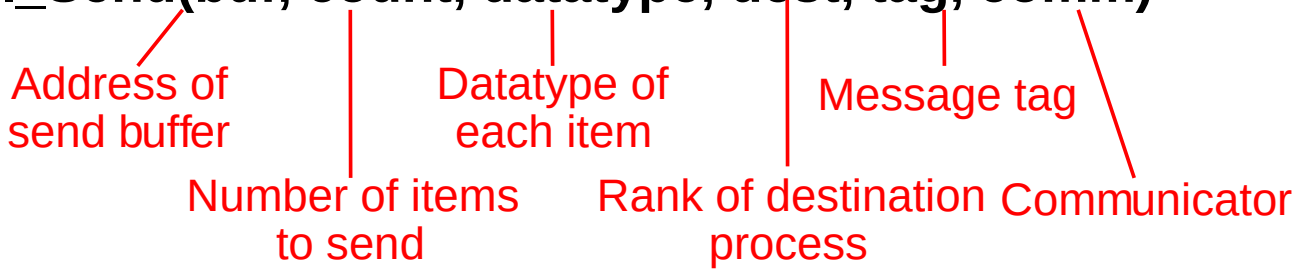
MPI (блокирующий) Send

Обобщенная форма:

MPI_SEND (start, count, datatype, dest, tag, comm)

- Буфер сообщения описывается как (**start, count, datatype**).
- Процесс получатель (**dest)** задается номером (rank) в заданном коммуникаторе (**comm)** .
- По завершению функции буфер может быть использован.

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

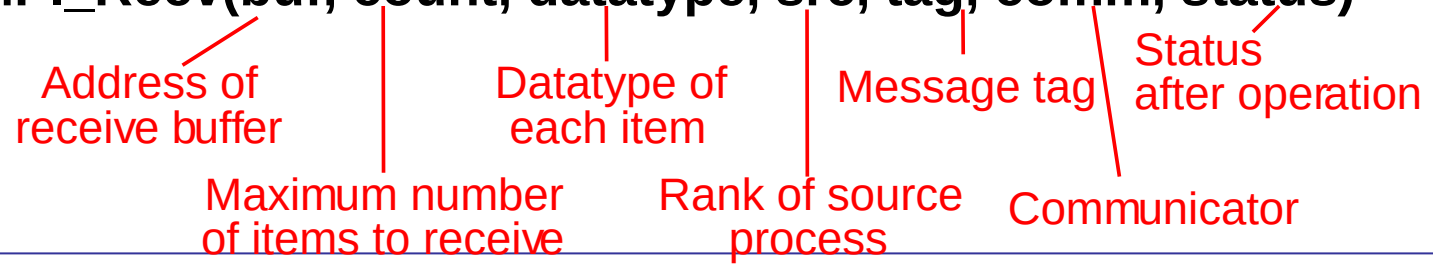


MPI (блокирующий) Receive

MPI_RECV(start, count, datatype, source, tag, comm, status)

- Ожидает, пока не придет соответствующее сообщение с заданными **`source`** и **`tag`**
- **`source`** – **номер процесса в коммуникаторе** **`comm`** или **`MPI_ANY_SOURCE`**.
- **`status`** содержит дополнительную информацию

MPI_Recv(buf, count, datatype, src, tag, comm, status)



MPI_Send

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

buf		адрес буфера
count	-	число пересылаемых элементов
Datatype	-	MPI datatype
dest	-	rank процесса-получателя
tag	-	определяемый пользователем параметр,
comm	-	MPI-коммуникатор

Пример:

```
MPI_Send(data,500,MPI_FLOAT,6,33,MPI_COMM_WORLD)
```

http://www-unix.mcs.anl.gov/mpi/www/www3/MPI_Send.html

MPI_Recv

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

buf		адрес буфера
count	-	число пересылаемых элементов
Datatype	-	MPI datatype
source	-	rank процесса-отправителя
tag	-	определяемый пользователем параметр,
comm	-	MPI-коммуникатор,
status	-	статус

Пример:

MPI_Send(data,500,MPI_FLOAT,6,33,MPI_COMM_WORLD)

http://www-unix.mcs.anl.gov/mpi/www/www3/MPI_Send.html

Пример: MPI Send/Receive (1)

```
#include <mpi.h>
#include <stdio.h>
int main(int argc, char *argv[]){
int numtasks, rank, dest, source, rc, tag=1;
char inmsg, outmsg='X';
MPI_Status Stat;
MPI_Init (&argc,&argv);
MPI_Comm_size (MPI_COMM_WORLD, &numtasks);
MPI_Comm_rank (MPI_COMM_WORLD, &rank);
if (rank == 0) {
  dest = 1;
rc = MPI_Send (&outmsg, 1, MPI_CHAR, dest, tag, MPI_COMM_WORLD);
  printf("Rank0 sent: %c\n", outmsg);
source = 1;
  rc = MPI_Recv (&inmsg, 1, MPI_CHAR, source, tag, MPI_COMM_WORLD,
     &Stat);  }
```

Пример: MPI Send/Receive (2)

```
else if (rank == 1) {
  source = 0;
  rc = MPI_Recv (&inmsg, 1, MPI_CHAR, source, tag, MPI_COMM_WORLD, &Stat);
  printf("Rank1 received: %c\n", inmsg);
 dest = 0;
  rc = MPI_Send (&outmsg, 1, MPI_CHAR, dest, tag, MPI_COMM_WORLD);
  }

MPI_Finalize();
}
```

Wildcarding (джокеры)

- Получатель может использовать джокер

  для получения сообщения от **ЛЮБОГО** процесса
  `MPI_ANY_SOURCE`

- Для получения сообщения с ЛЮБЫМ тэгом
  `MPI_ANY_TAG`

- Реальные номер процесса-отправителя и тэг возвращаются через параметр ***`status`***

Информация о завершившемся приеме сообщения

- Возвращается функцией **`MPI_Recv`** через `параметр` **`status`**

- Содержит:
  - Source: ***`status.MPI_SOURCE`***
  - Tag: ***`status.MPI_TAG`***
  - Count: ***`MPI_Get_count`***

Полученное сообщение

- Может быть меньшего размера, чем указано в функции MPI_Recv
- **`count`** – число реально полученных элементов

C:

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

Пример

```
int recvd_tag, recvd_from, recvd_count;
MPI_Status status;
MPI_Recv (..., MPI_ANY_SOURCE, MPI_ANY_TAG, ..., &status )
recvd_tag  = status.MPI_TAG;
recvd_from = status.MPI_SOURCE;
MPI_Get_count( &status, datatype, &recvd_count );
```

Условия успешного взаимодействия «точка-точка»

- Отправитель должен указать правильный rank получателя
- Получатель должен указать верный rank отправителя
- Одинаковый коммуникатор
- Тэги должны соответствовать друг другу
- Буфер у процесса-получателя должен быть достаточного объема

MPI_Probe

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

Проверка статуса операции приема сообщения.

Параметры аналогичны функции MPI_Recv

Пример

```
if (rank == 0) {
// Send size  of integers to process 1
    MPI_Send(buf, size, MPI_INT, 1, 0,
      MPI_COMM_WORLD);
    printf("0 sent %d numbers to 1\n",
      size);
 } else if (rank == 1) {
    MPI_Status status;
    // Probe for an incoming message from
      process
    MPI_Probe (0, 0, MPI_COMM_WORLD,
      &status);
    MPI_Get_count (&status, MPI_INT,
      &size);
```

```
int* number_buf =
(int*)malloc(sizeof(int) * size);
 // Now receive the message with the
allocated buffer
 MPI_Recv (number_buf, size, MPI_INT,
                0, 0, MPI_COMM_WORLD,
                 MPI_STATUS_IGNORE);
printf("1 dynamically received %d
numbers from 0.\n",
          number_amount);
    free(number_buf);
 }
```

Совмещение передач типа «отсылка-прием» (1)

- Функция MPI_Sendrecv совмещает выполнение операций передачи и приема.
- Обе операции используют один и тот же коммуникатор, но идентификаторы сообщений могут различаться.
- Расположение в адресном пространстве процесса принимаемых и передаваемых данных не должно пересекаться.
- Пересылаемые (по send и recv) данные могут быть различного типа и иметь разную длину.

Совмещение передач типа «отсылка-прием» (2)

```
int MPI_Sendrecv (void *sendbuf,
        int sendcount, MPI_Datatype sendtype,
        int dest, int sendtag,
        void *rcvbuf, int rcvcount, MPI_Datatype rcvtype,
         int source, int rcvtag,
        MPI_Comm comm,
        MPI_Status *status)
```

Обмен данными одного типа с замещением

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 коммуникации

Цель – уменьшение времени работы параллельной программы за счет совмещения вычислений и обменов.

Неблокирующие операции завершаются, не дожидаясь окончания передачи данных. В отличие от аналогичных блокирующих функций изменен критерий завершения операций – немедленное завершение.

Проверка состояния передач и ожидание завершение передач выполняются специальными функциями.

Форматы неблокирующих функций

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

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

Проверка завершения операций **`MPI_Wait()`** and **`MPI_Test()`.**

`MPI_Wait()` ожидание завершения.

`MPI_Test()` проверка завершения. Возвращается флаг, указывающий на результат завершения.

Неблокирующий send

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

“_I” - “Immediate” немедленный возврат

Неблокирующий receive

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

Wait/Test функции

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

Множественные проверки

- Test или wait для завершения одной (и только одной) передачи:
  - int MPI_Waitany (…)
  - int MPI_Testany (…)
- Test или wait завершения всех передач:
  - int MPI_Waitall (…)
  - int MPI_Testall (…)
- Test или wait завершения всех возможных к данному моменту:
  - int MPI_Waitsome(…)
  - int MPI_Testsome(…)

Пример использования асинхронных передач

```
MPI_Comm_rank(MPI_COMM_WORLD, &myrank);    /* find rank */
if (myrank == 0) {
    int x;
    MPI_Isend (&x,1,MPI_INT, 1, msgtag, MPI_COMM_WORLD, &req1);
    compute();
    MPI_Wait (&req1, &status);
} else if (myrank == 1) {
    int x;
    MPI_Recv (&x,1,MPI_INT,0,msgtag, MPI_COMM_WORLD, &status);
}
```

Коллективные передачи

- Передача сообщений между группой процессов
- Вызываются ВСЕМИ процессами в коммуникаторе
- Примеры:
  - Broadcast, scatter, gather (рассылка данных)
  - Global sum, global maximum, и т.д. (Коллективные операции)
  - Барьерная синхронизация

Функции коллективных передач

Collective Communication Routines		
MPI_Allgather	MPI_Allgatherv	MPI_Allreduce
MPI_Alltoall	MPI_Alltoallv	MPI_Barrier
MPI_Bcast	MPI_Gather	MPI_Gatherv
MPI_Op_create	MPI_Op_free	MPI_Reduce
MPI_Reduce_scatter	MPI_Scan	MPI_Scatter
MPI_Scatterv		

Характеристики коллективных передач

- Коллективные операции не являются помехой операциям типа «точка-точка» и наоборот
- Все процессы коммуникатора должны вызывать коллективную операцию
- Синхронизация не гарантируется (за исключением барьера)
- Нет неблокирующих коллективных операций
- Нет тэгов
- Принимающий буфер должен точно соответствовать размеру отсылаемого буфера

Барьерная синхронизация

- Приостановка процессов до выхода ВСЕХ процессов коммуникатора в заданную точку синхронизации

```
int MPI_Barrier (MPI_Comm comm)
```

Широковещательная рассылка

- One-to-all передача: один и тот же буфер отсылается от процесса root всем остальным процессам в коммуникаторе
- ```
  int MPI_Bcast (void *buffer, int, count,
    MPI_Datatype datatype,int root, MPI_Comm comm)
  ```
- Все процессы должны указать один тот же root и communicator
  ```

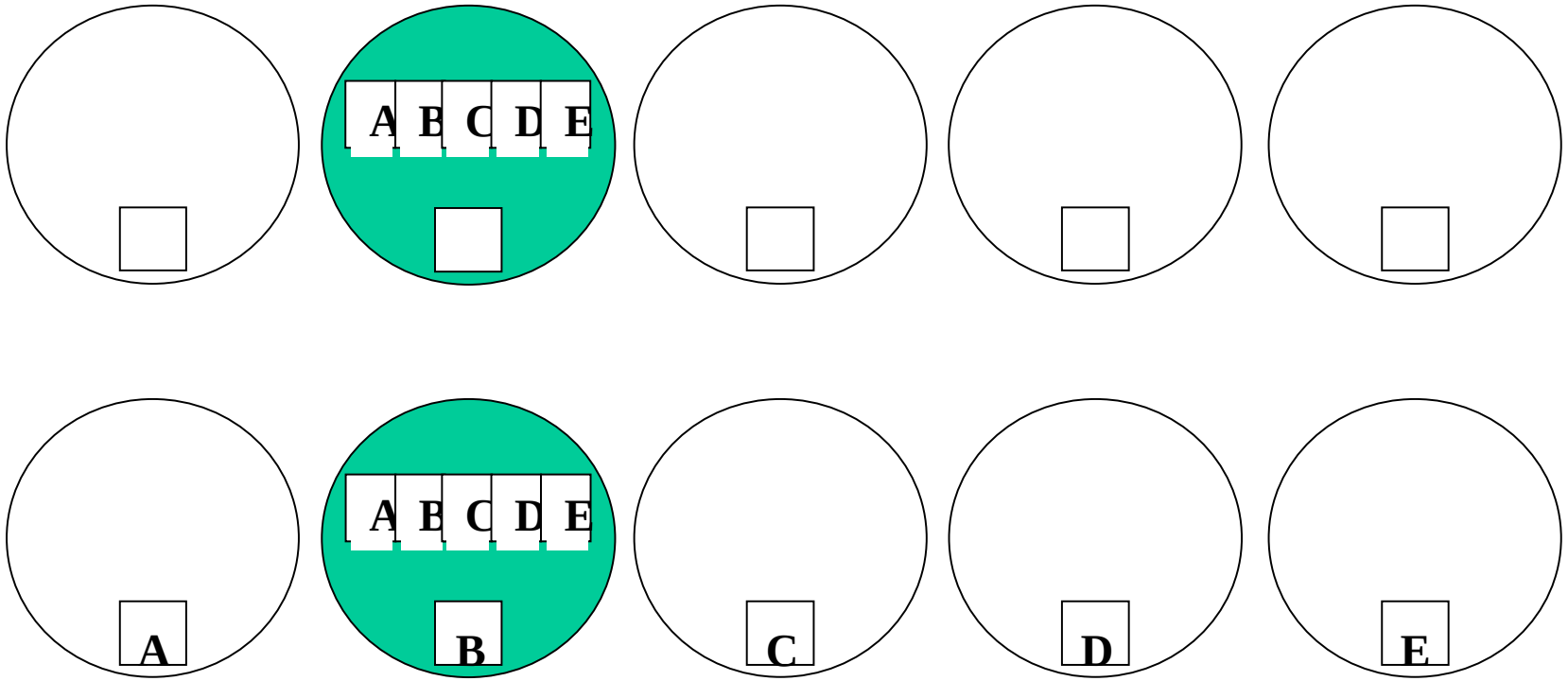
Scatter

- One-to-all communication: различные данные из одного процесса рассылаются всем процессам коммуникатора (в порядке их номеров)

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

- *sendcount* – число элементов, посланных каждому процессу, не общее число отосланных элементов;
- send параметры имеют смысл только для процесса root

Scatter – графическая иллюстрация

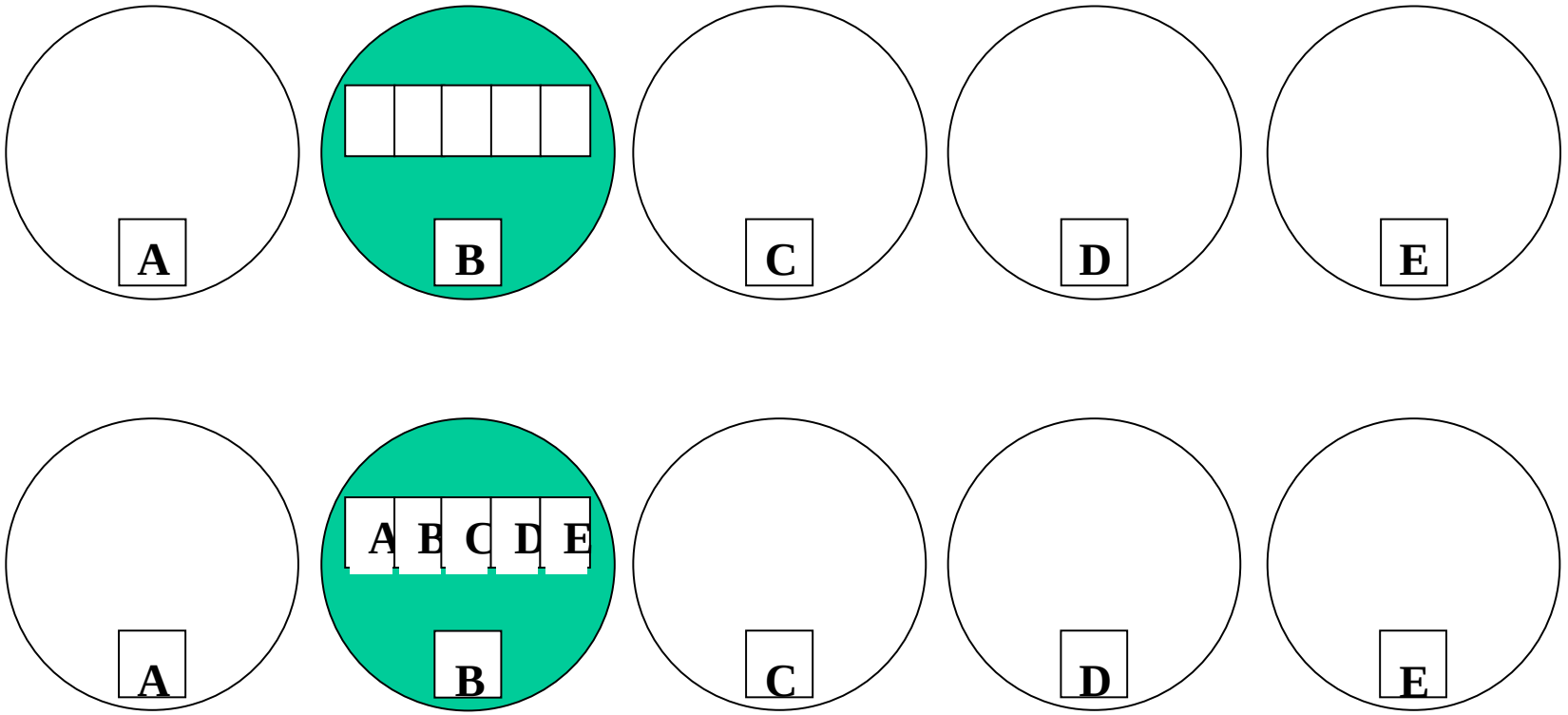


Gather

- All-to-one передачи: различные данные собираются процессом root
- Сбор данных выполняется в порядке номеров процессов
- Длина блоков предполагается одинаковой, т.е. данные, посланные процессом i из своего буфера sendbuf, помещаются в i-ю порцию буфера recvbuf процесса root. Длина массива, в который собираются данные, должна быть достаточной для их размещения.

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

Gather – графическая илюстрация



Глобальные операции редукции

- Операции выполняются над данными, распределенными по процессам коммуникатора
- Примеры:
  - Глобальная сумма или произведение
  - Глобальный максимум (минимум)
  - Глобальная операция, определенная пользователем

Общая форма

```
int MPI_Reduce(void* sendbuf, void* recvbuf,
int count, MPI_Datatype datatype, MPI_Op op,
int root,MPI_Comm comm)
```

- **count** число операций “*op*” выполняемых над последовательными элементами буфера **sendbuf**
- (также размер **recvbuf**)

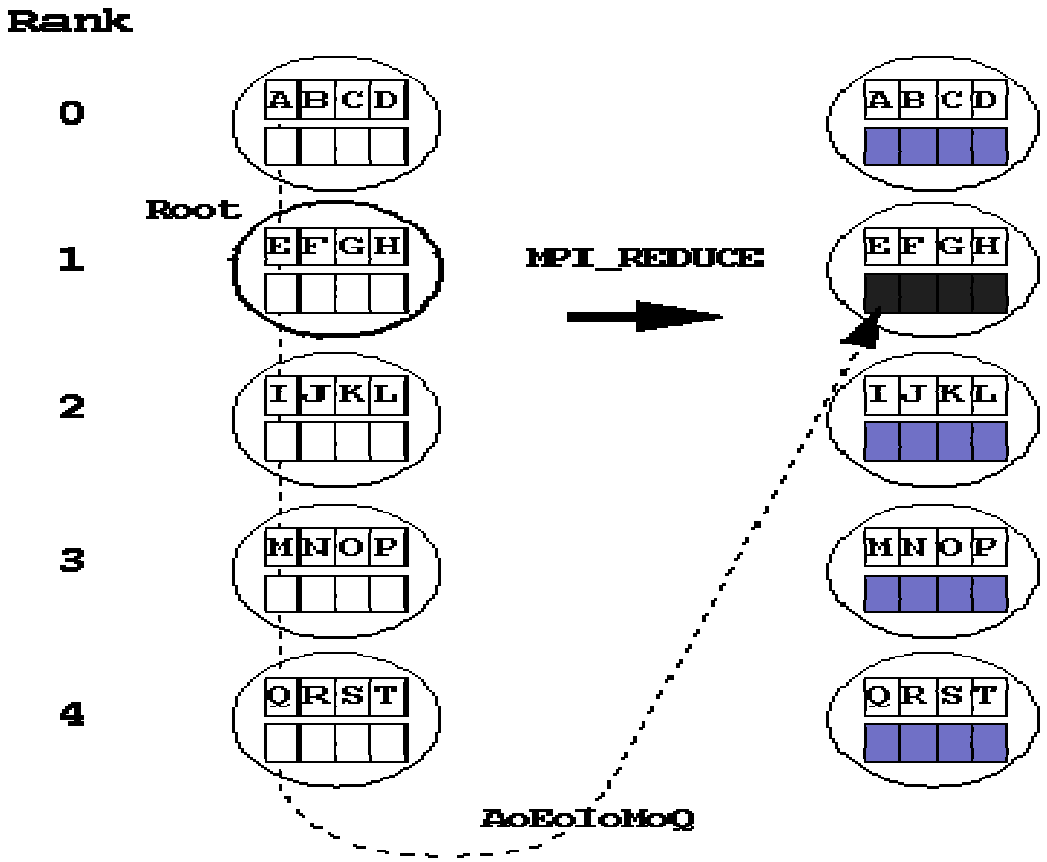
- **op** является ассоциативной операцией, которая выполняется над парой операндов типа **datatype**

  и возвращает результат того же типа

Предопределенные операции редукции

MPI Name	Function
MPI_MAX	Maximum
MPI_MIN	Minimum
MPI_SUM	Sum
MPI_PROD	Product
MPI_LAND	Logical AND
MPI_BAND	Bitwise AND
MPI_LOR	Logical OR
MPI_BOR	Bitwise OR
MPI_LXOR	Logical exclusive OR
MPI_BXOR	Bitwise exclusive OR
MPI_MAXLOC	Maximum and location
MPI_MINLOC	Minimum and location

MPI_Reduce


Rank
0
1
2
3
4
Root
A B C D
E F G H
I J K L
M N O P
Q R S T
MPI_REDUCE
A B C D
E F G H
I J K L
M N O P
Q R S T
AoEoIoMoQ

Варианты MPI_REDUCE

- MPI_ALLREDUCE - нет root процесса (все получают рез-т)

- MPI_REDUCE_SCATTER
- MPI_SCAN - “parallel prefix”

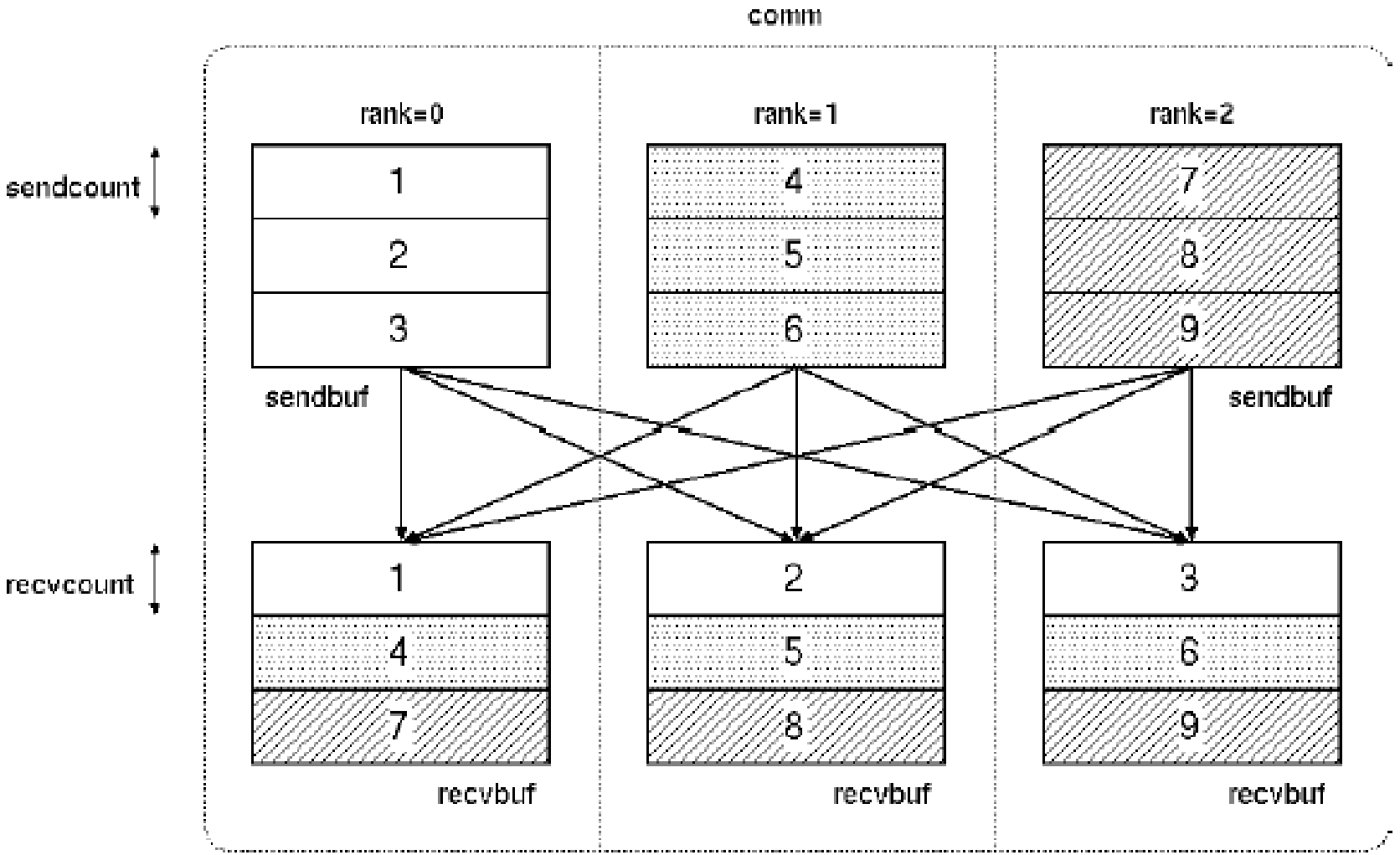
MPI_ALLTOALL

```
int MPI_Alltoall( void* sendbuf,
                  int sendcount,              /* in */
                  MPI_Datatype sendtype,      /* in */
                  void* recvbuf,              /* out */
                  int recvcount,              /* in */
                  MPI_Datatype recvtype,      /* in */
                  MPI_Comm comm);
```

Описание:

- Рассылка сообщений от каждого процесса каждому
- j-ый блок данных из процесса i принимается j-ым процессом и размещается в i-ом блоке буфера recvbuf

MPI ALLTOALL



Пример использования MPI_ALLTOALL

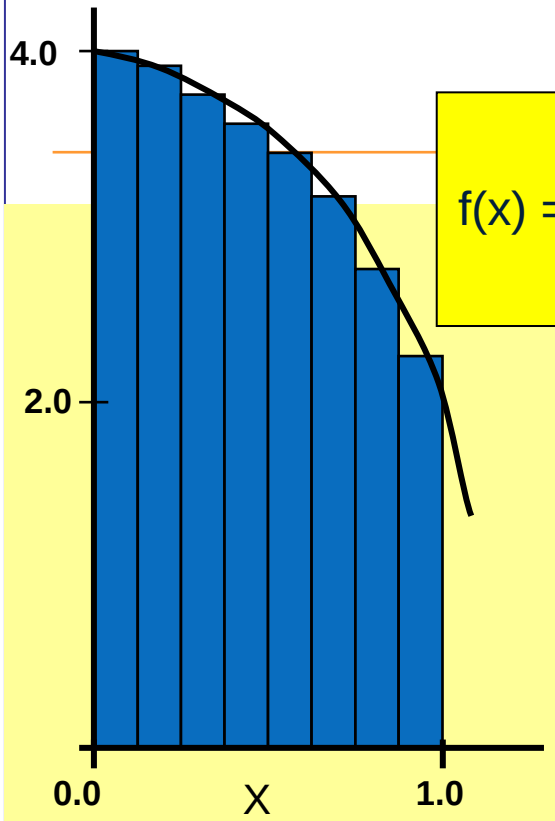
```
#include “mpi.h”
int main( int argc, char* argv[] )
{
    int i;
    int rank, nproc;
    int isend[3], irecv[3];
    MPI_Init( &argc, &argv );
  MPI_Comm_size( MPI_COMM_WORLD,
&nproc );
  MPI_Comm_rank( MPI_COMM_WORLD,
&rank );
    for(i=0; i<nproc; i++)
        isend[i] = i + nproc * rank;
```

```
MPI_Alltoall(isend, 1,
MPI_INTEGER, irecv, 1,
MPI_INTEGER,

MPI_COMM_WORLD);
    for(i=0; i<3; i++)
        printf(“irecv = %d\n”,
irecv[i]);

    MPI_Finalize();
}
```

Пример: параллельное численное интегрирование



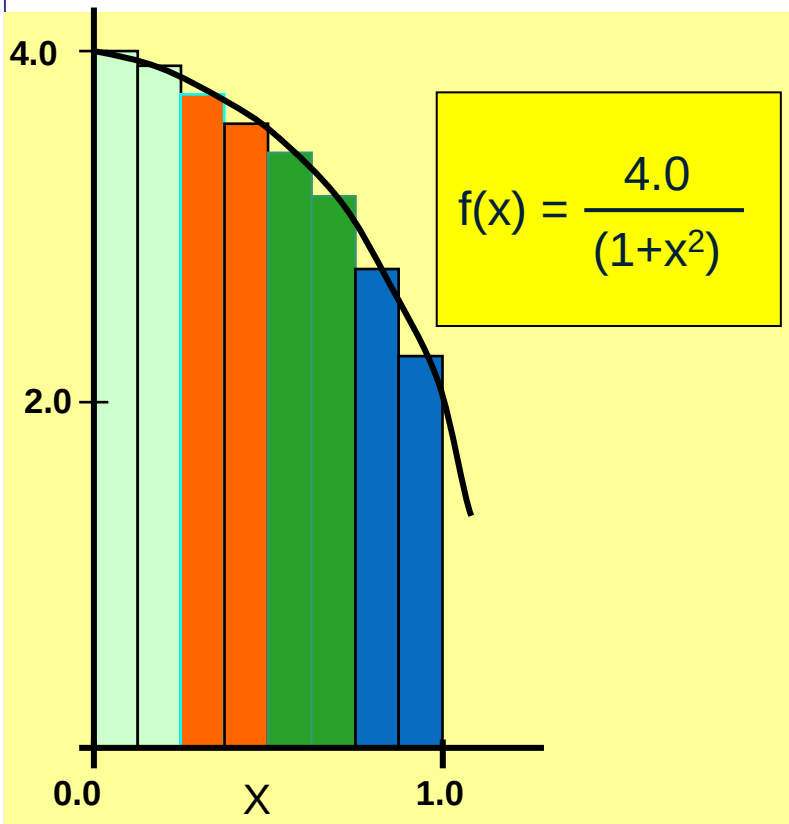
$$f(x) = \frac{4.0}{(1+x^2)}$$

```
static long num_steps=100000;
 double step, pi;

void main()
{  int i;
   double x, sum = 0.0;

   step = 1.0/(double) num_steps;
   for (i=0; i< num_steps; i++){
      x = (i+0.5)*step;
      sum = sum + 4.0/(1.0 + x*x);
   }
   pi = step * sum;
   printf("Pi = %f\n",pi);
}
```

Пример: параллельное численное интегрирование




```
static long num_steps=100000; double
step, pi;

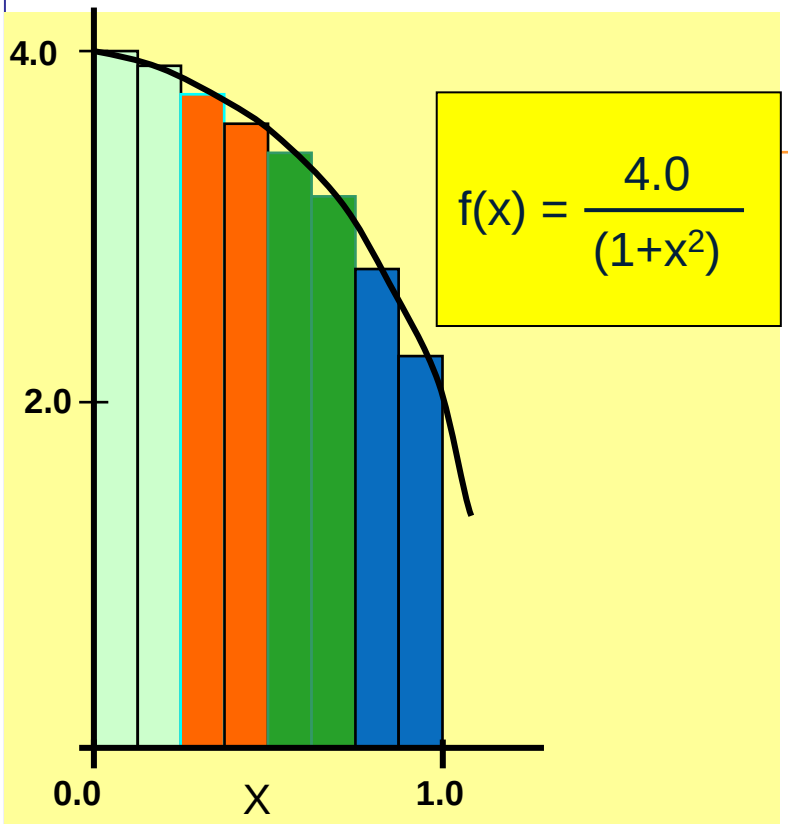
void int main(int argc, char *argv[]){
{  int i;
   double x, sum = 0.0;
  int nProc, myRank;
  int i, N;
  double partialSum, totalSum;
  MPI_Status status;
  MPI_Init(&argc, &argv);

  MPI_Comm_rank(MPI_COMM_WORLD, &myRank);
  MPI_Comm_size(MPI_COMM_WORLD, &nProc);

  if (argc != 2){
    printf("Error: Accuracy is
required\n");
  }

  if (myRank == 0){
    N = (int)(2*atof(argv[1]));
  }
  MPI_Bcast(&N, 1, MPI_INT, 0,
MPI_COMM_WORLD);
```

Пример: параллельное численное интегрирование



$$f(x) = \frac{4.0}{(1+x^2)}$$

P0

```
partialSum = 0.0;
  for (i=N-myRank;i>=0;i-=nProc){
    partialSum += pow(-1.0, i)/(2*i +
1);
  }

  printf("Partial Sum from %d is %e\n",
myRank, partialSum);

  MPI_Reduce(&partialSum, &totalSum, 1,
MPI_DOUBLE,
             MPI_SUM, 0,
MPI_COMM_WORLD);

  if (myRank == 0){
    printf("pi = %e\n", totalSum * 4);
  }

  MPI_Finalize();
  return(0);
}
```

Пример: вычисление PI (1)

```
#include "mpi.h"
#include <math.h>
int main(int argc, char *argv[])
{
   int done = 0, n, myid, numprocs, i, rc;
   double PI25DT = 3.14159265358979323846264643;
   double mypi, pi, h, sum, x, a;
   MPI_Init(&argc,&argv);
   MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
   MPI_Comm_rank(MPI_COMM_WORLD,&myid);
   while (!done)  {
     if (myid == 0) {
       printf("Enter the number of intervals: (0 quits) ");
       scanf("%d",&n);}
     MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD);
     if (n == 0) break;
```

Пример: PI (2)

```
    h   = 1.0 / (double) n;
    sum = 0.0;
    for (i = myid + 1; i <= n; i += numprocs) {
      x = h * ((double)i - 0.5);
      sum += 4.0 / (1.0 + x*x);
    }
    mypi = h * sum;
    MPI_Reduce(&mypi, &pi, 1, MPI_DOUBLE, MPI_SUM, 0,
               MPI_COMM_WORLD);
    if (myid == 0)
      printf("pi is approximately %.16f, Error is %.16f\n",
             pi, fabs(pi - PI25DT));
  }
  MPI_Finalize();
 return 0;}
```