MPI

A.Klypin

- MPI tasks
- Simple Example
- Collective communications
- Point-to-Point Communications

MPI

- The same code is run on many processors
- Each MPI task can use many OpenMP threads. So, a task is not necessarily mapped on a core or a processor, but often it does.
- After initialization each MPI task gets its unique id (rank)
- All MPI tasks are equal. For programming purposes it is convenient to name the rank=0 task as root and use it differently.
- Exchange of data between tasks is done by library calls.
- Semantics for Fortran and C are very similar.
- Root (and some other tasks) can be allocated to different compute nodes with larger memory.
- Submit a PBS script to a queue. The script gives details of your job and has a line mpiexec -np NNN mycode.exe (or mpirun) where -np NNN specifies the number of MPI tasks.

Simple Example

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

int main(argc,argv)
int argc;
char *argv[]; {
int  numtasks, rank, rc;

rc = MPI_Init(&argc,&argv);
if (rc != MPI_SUCCESS) {
  printf ("Error starting MPI program. Terminating.\n");
  MPI_Abort(MPI_COMM_WORLD, rc);
  }

MPI_Comm_size(MPI_COMM_WORLD,&numtasks);
MPI_Comm_rank(MPI_COMM_WORLD,&rank);
printf ("Number of tasks= %d My rank= %d\n", numtasks,rank);

/*******  do some work *******/

MPI_Finalize();
}
```

```fortran
   program simple
   include 'mpif.h'

   integer numtasks, rank, ierr, rc

   call MPI_INIT(ierr)
   if (ierr .ne. MPI_SUCCESS) then
      print *,'Error starting MPI program. Terminating.'
      call MPI_ABORT(MPI_COMM_WORLD, rc, ierr)
   end if

   call MPI_COMM_RANK(MPI_COMM_WORLD, rank, ierr)
   call MPI_COMM_SIZE(MPI_COMM_WORLD, numtasks, ierr)
   print *, 'Number of tasks=',numtasks,' My rank=',rank

C ****** do some work ******

   call MPI_FINALIZE(ierr)

   end
```

https://computing.llnl.gov/tutorials/mpi/#Getting_Started

Collective Communications

- Collective communication must involve **all** processes
- Types of communications:
    - Synchronization
    - Data transfer : broadcast, scatter gather, all-to-all
    - Collective computation: (reductions) - one member of the group collects data from the other members and performs an operation (min, max, add, multiply, etc.) on that data.

```
CALL MPI_INIT(ierr)       ! Initialize MPI
CALL MPI_COMM_RANK(MPI_COMM_WORLD,rank,ierr)
CALL MPI_COMM_SIZE(MPI_COMM_WORLD, numprocs,ierr)

  If(rank == 0) Then
      CALL InitValues (SCALEL)
      CALL Bcast_InitValues(SCALEL)
  Else
      CALL Bcast_InitValues(SCALEL)
  EndIf

------------------------------------------------------------
  SUBROUTINE InitValues (SCALEL)
      Write (*, '(A,$)') 'Enter ScaleLength ='
      READ ( *, * ) SCALEL
  End SUBROUTINE InitValues
```

```
!------------------------------------------------------------
  SUBROUTINE Bcast_InitValues (SCALEL)
      Nseed = 12312
      AMPLT = 1.123
    CALL MPI_Bcast(SCALEL,1,MPI_REAL,0,MPI_COMM_WORLD,ierr)
    CALL MPI_Bcast(Nseed,1,MPI_INTEGER,0,MPI_COMM_WORLD,ierr)
    CALL MPI_Bcast(AMPLT,1,MPI_REAL,0,MPI_COMM_WORLD,ierr)

  End SUBROUTINE Bcast_InitValues
```

- One task reads input from screen and distributes it to all others

https://computing.llnl.gov/tutorials/mpi/#Getting_Started

Simple Example: matrix transposition

```
!-------------------------------------------
!        Matrix transposition
!-------------------------------------------
Module Struc
  integer, parameter :: NROW = 16
  integer, dimension(NROW,NROW) :: G,Gb
  integer :: rank
Contains
!-------------------------------------------
SUBROUTINE Transpose
use mpi
  CALL MPI_ALLtoALL(G ,NROW,MPI_INT, &
                    Gb,NROW,MPI_INT, &
               MPI_COMM_WORLD,ierr)
end SUBROUTINE Transpose

end module STRUC
```

Three-dimensional matrix **A(Nrow,Nrow,Nrow)** is split such that each task **k** has its one page **G(:,:) = A(:,:,k)**
After transposition **Gb(:,:) = A(k;:,:)**

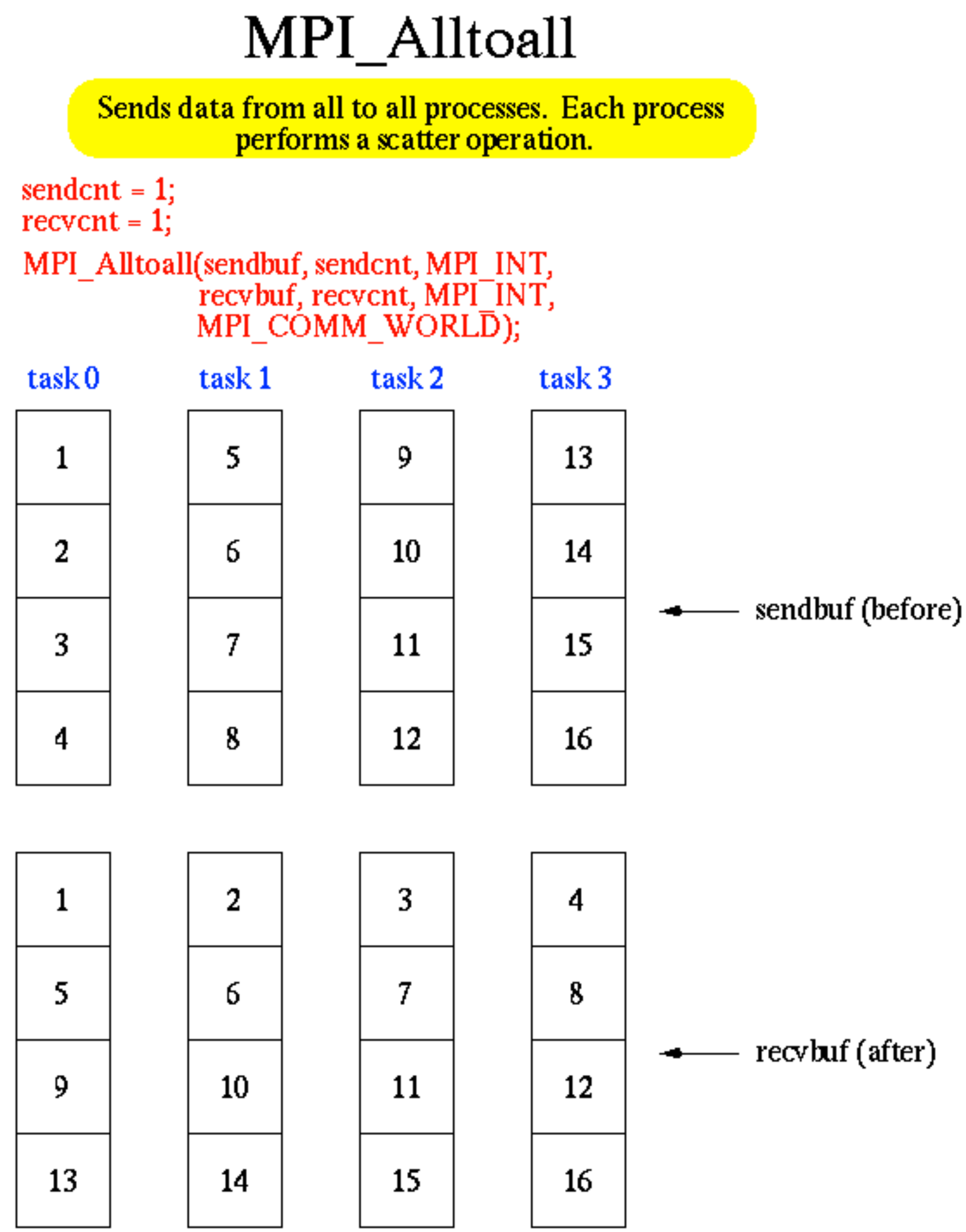
```
!-------------------------------------------
Program MatTran
use Struc
use mpi
integer*8 :: nLevel(10),ii

  CALL MPI_INIT(ierr)
  CALL MPI_COMM_RANK(MPI_COMM_WORLD,rank,ierr)
  CALL MPI_COMM_SIZE(MPI_COMM_WORLD, numprocs,ierr)

  k = rank +1
  DO j=1,NROW   ! fill the maxtrix
  DO i=1,NROW
    G(i,j) = i + j*1000 +k*1000000
  EndDo
  EndDo
nLevel = 0
If(rank == 0)
   do i =1,10
     nLevel(i) =2_8**(i+1)
   enddo
EndIf

CALL MPI_Bcast(nLevel,10,MPI_LONG,0,MPI_COMM_WORLD,ierr)
CALL Transpose

  CALL MPI_Barrier(MPI_COMM_WORLD,ierr)
  CALL MPI_FINALIZE(ierr)
```

MPI_AlltoALL

https://computing.llnl.gov/tutorials/mpi/#Getting_Started

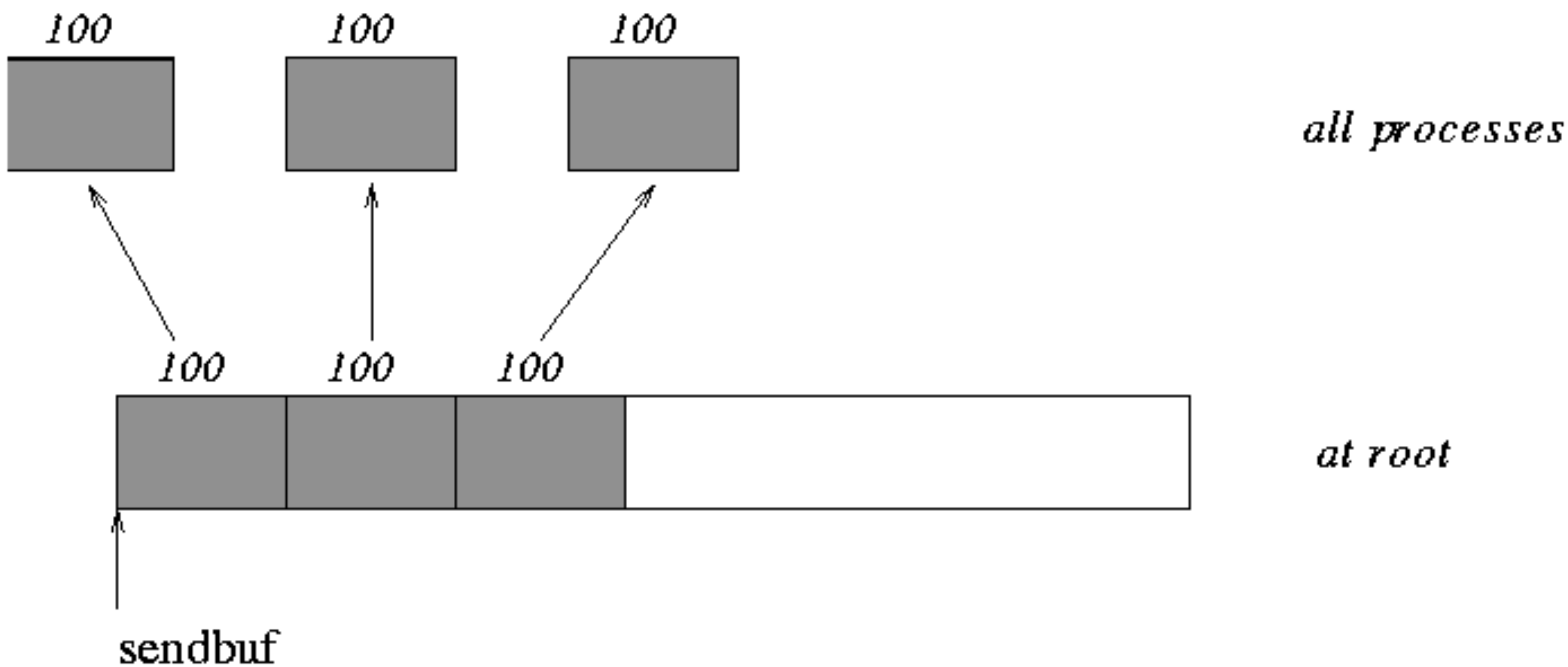
sendbuf = starting address of send buffer
recvbuf = address of receive buffer
sebdcnt = number of elements send to each process
recvcnt = number of elements received

j-th block from process **i** is received by process **j** and placed in the **i**-th block of recvbuf



MPI_Scatter: root distributes data

```
MPI_Comm comm;
    int gsize,*sendbuf;
    int root, rbuf[100];
    ...
    MPI_Comm_size( comm, &gsize);
    sendbuf = (int *)malloc(gsize*100*sizeof(int));
    ...
    MPI_Scatter( sendbuf, 100, MPI_INT, rbuf, 100, MPI_INT, root, comm);
```

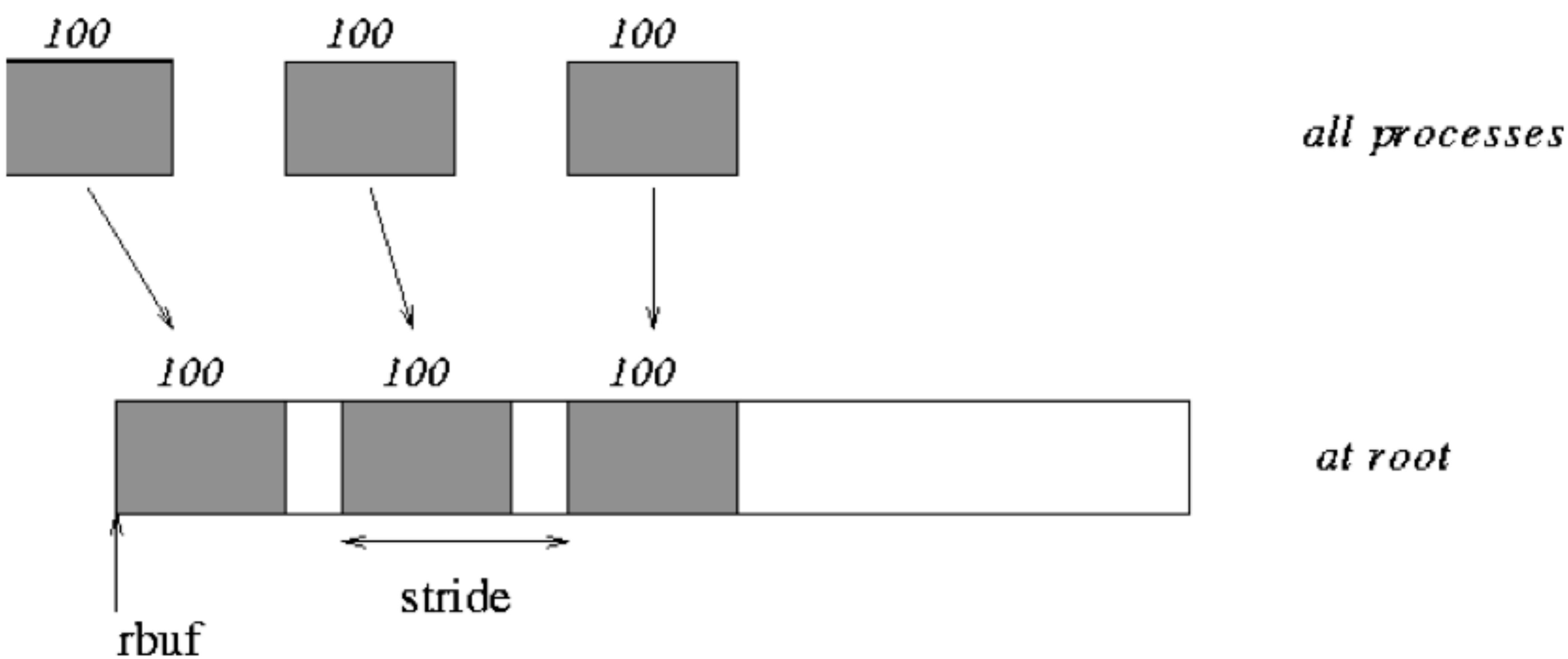


http://www.mpi-forum.org/docs/mpi-11-html/node72.html

Root gathers data from other tasks

```
MPI_Comm_size( comm, &gsize);
    rbuf = (int *)malloc(gsize*stride*sizeof(int));
    displs = (int *)malloc(gsize*sizeof(int));
    rcounts = (int *)malloc(gsize*sizeof(int));
    for (i=0; i<gsize; ++i) {
        displs[i] = i*stride;
        rcounts[i] = 100;
    }
    MPI_Gatherv( sendarray, 100, MPI_INT, rbuf, rcounts, displs, MPI_INT,
                                                        root, comm);
```

Note that the program is erroneous if *stride* < *100*.



Point-to-Point communications

MPI_SEND: send message to task ***dest***

MPI_RECV: receive message from task ***dest***

C synopsis

```
#include <mpi.h>
int MPI_Send(void* buf,int count,MPI_Datatype datatype,
    int dest,int tag,MPI_Comm comm);
```

C++ synopsis

```
#include mpi.h
void MPI::Comm::Send(const void* buf, int count, const MPI::Datatype& datatype,
                int dest, int tag) const;
```

FORTRAN synopsis

```
include 'mpif.h' or use mpi
MPI_SEND(CHOICE BUF,INTEGER COUNT,INTEGER DATATYPE,INTEGER DEST,
     INTEGER TAG,INTEGER COMM,INTEGER IERROR)
```

buf
: The initial address of the send buffer (choice) (IN)

count
: The number of elements in the send buffer (non-negative integer) (IN)

datatype
: The data type of each send buffer element (handle) (IN)

dest
: The rank of the destination task in *comm*(integer) (IN)

tag
: The message tag (positive integer) (IN)

comm
: The communicator (handle) (IN)

IERROR
: The FORTRAN return code. It is always the last argument.

Example of Point-to-Point communications

```
if (my_rank ==0) {
  fputs(greeting, stdout);
  for (partner = 1; partner < size; partner++){

    MPI_Recv(greeting, sizeof(greeting), MPI_BYTE, partner, 1, MPI_COMM_WORLD, &stat);
    fputs (greeting, stdout);

  }
}
else {
  MPI_Send(greeting, strlen(greeting)+1, MPI_BYTE, 0,1,MPI_COMM_WORLD);
}
```