



DATA SCIENCE &
SCIENTIFIC COMPUTING



i o m

Istituto Officina
dei Materiali

exact

L05: MPI programming (2)

- Stefano Cozzini
- CNR-IOM and eXact lab srl

Collective operations

- Collective routines provide a higher-level way to organize a parallel program
- Each process executes the same communication operations
- MPI provides a rich set of collective operations...

Collective operations

- Communications involving group of processes in a communicator.
- Groups and communicators can be constructed “by hand” or using topology routines.
- Tags are not used; different communicators deliver similar functionality.
- No non-blocking collective operations.
- Three classes of operations: synchronization, data movement, collective computation.

MPI_Barrier

Stop processes until all processes within a communicator reach the barrier

Almost never required in a parallel program Occasionally useful in measuring performance and load balancing

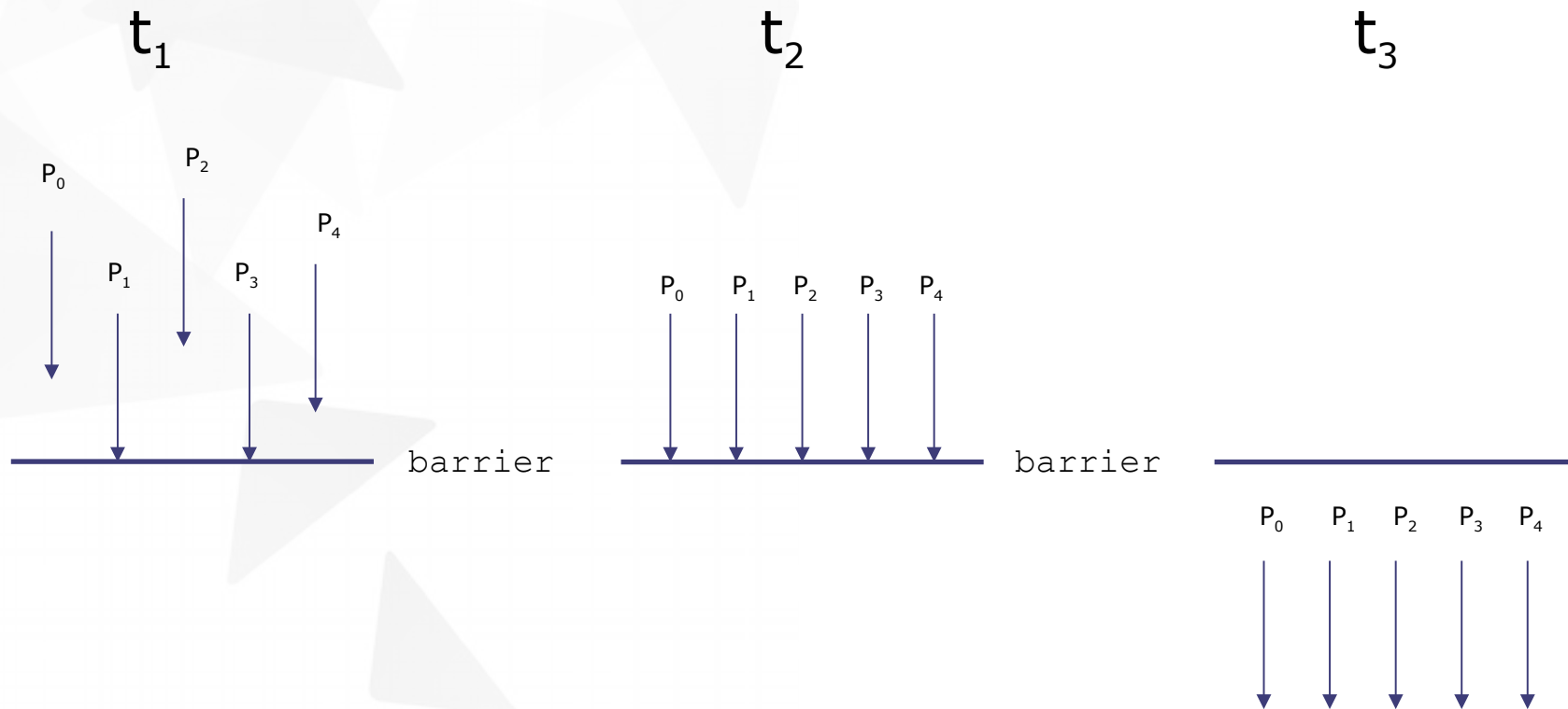
Fortran:

```
CALL MPI_BARRIER( comm, ierr)
```

C:

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

Barrier



Broadcast (MPI_BCAST)

One-to-all communication: same data sent from root process to all others in the communicator

Fortran:

```
INTEGER count, type, root, comm, ierr  
CALL MPI_BCAST(buf, count, type, root, comm, ierr)
```

Buf array of type **type**

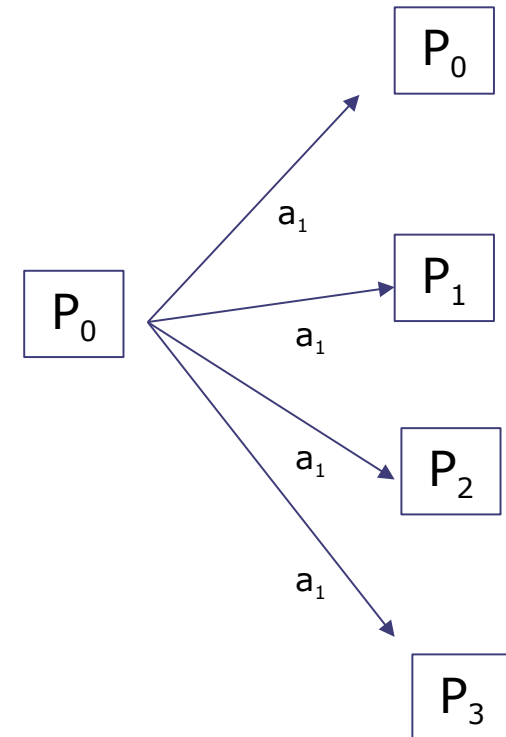
C:

```
int MPI_Bcast(void *buf, int count, MPI_Datatype  
             datatype, int root, MPI_Comm comm)
```

All processes must specify same **root**, **rank** and **comm**

Broadcast

```
PROGRAM broad_cast
  INCLUDE 'mpif.h'
  INTEGER ierr, myid, nproc, root
  INTEGER status(MPI_STATUS_SIZE)
  REAL A(2)
  CALL MPI_INIT(ierr)
  CALL MPI_COMM_SIZE(MPI_COMM_WORLD, nproc, ierr)
  CALL MPI_COMM_RANK(MPI_COMM_WORLD, myid, ierr)
  root = 0
  IF( myid .EQ. 0 ) THEN
    a(1) = 2.0
    a(2) = 4.0
  END IF
  CALL MPI_BCAST(a, 2, MPI_REAL, 0, MPI_COMM_WORLD, ierr)
  WRITE(6,*) myid, ': a(1)=', a(1), 'a(2)=', a(2)
  CALL MPI_FINALIZE(ierr)
END
```



Reduction

The reduction operation allow to:

- Collect data from each process
- Reduce the data to a single value
- Store the result on the root processes
- Store the result on all processes

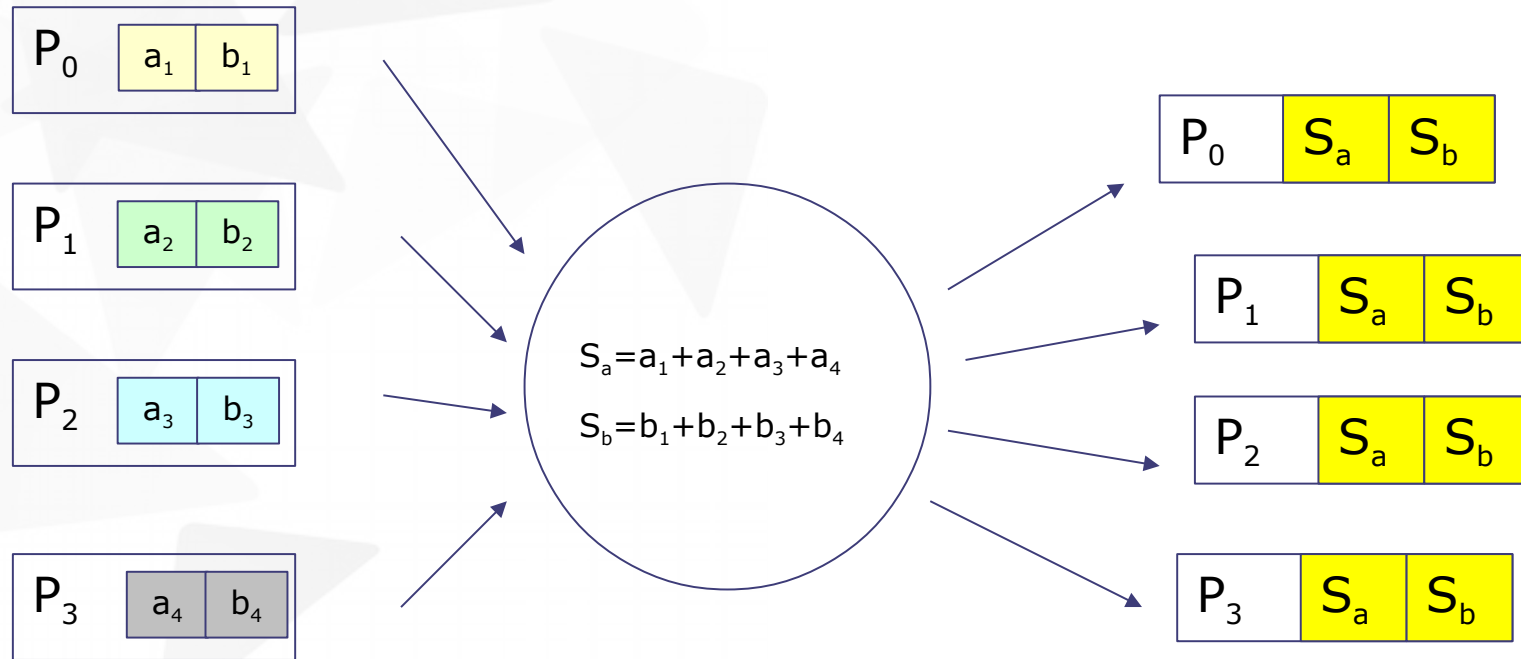
MPI_Reduce and MPI_Allreduce

C:

```
int MPI_Reduce(void * snd_buf, void * rcv_buf, int count,  
              MPI_Datatype type, MPI_Op op, int root, MPI_Comm comm)
```

```
int MPI_Allreduce(void * snd_buf, void * rcv_buf, int count,  
                 MPI_Datatype type, MPI_Op op, MPI_Comm comm)
```

Reduce, Parallel Sum



Reduction function works with arrays

other operation: product, min, max, and,

Internally is usually implemented with a binary tree

MPI_REDUCE and MPI_ALLREDUCE

Fortran:

MPI_REDUCE(snd_buf,rcv_buf,count,type,op,root,comm,ierr)

snd_buf input array of type type containing local values.
rcv_buf output array of type type containing global results
count (INTEGER) number of element of snd_buf and rcv_buf
type (INTEGER) MPI type of snd_buf and rcv_buf
op (INTEGER) parallel operation to be performed
root (INTEGER) MPI id of the process storing the result
comm (INTEGER) communicator of processes involved in the operation
ierr (INTEGER) output, error code (if ierr=0 no error occurs)

MPI_ALLREDUCE(snd_buf,rcv_buf,count,type,op,comm,ierr)

The argument root is missing, the result is stored to all processes.

Predefined Reduction Operations

| MPI op | 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 |

Reduce, example

```
PROGRAM reduce
  INCLUDE 'mpif.h'
  INTEGER ierr, myid, nproc, root
  INTEGER status(MPI_STATUS_SIZE)
  REAL A(2), res(2)
  CALL MPI_INIT(ierr)
  CALL MPI_COMM_SIZE(MPI_COMM_WORLD, nproc, ierr)
  CALL MPI_COMM_RANK(MPI_COMM_WORLD, myid, ierr)
  root = 0
  a(1) = 2.0
  a(2) = 4.0
  CALL MPI_REDUCE(a, res, 2, MPI_REAL, MPI_SUM, root,
& MPI_COMM_WORLD, ierr)
  IF( myid .EQ. 0 ) THEN
    WRITE(6,*) myid, ': res(1)=', res(1), 'res(2)=', res(2)
  END IF
  CALL MPI_FINALIZE(ierr)
END
```

MPI_Scatter

One-to-all communication: different data sent from root process to all others in the communicator

Fortran:

```
CALL MPI_SCATTER(sender sndbuf, sndcount, sndtype, receiver rcvbuf, rcvcount, rcvtype,  
root, comm, ierr)
```

Arguments definition are like other MPI subroutine

sndcount is the number of elements sent to each process, not the size of **sndbuf**, that should be **sndcount** times the number of process in the communicator

The sender arguments are significant only at root

MPI_Gather

One-to-all communication: different data collected by the root process, from all other processes in the communicator. Is the opposite of Scatter

Fortran:

```
CALL MPI_GATHER(sndbuf, sendersndcount, sndtype, rcvbuf, receiverrcvcount, rcvtype,  
               root, comm, ierr)
```

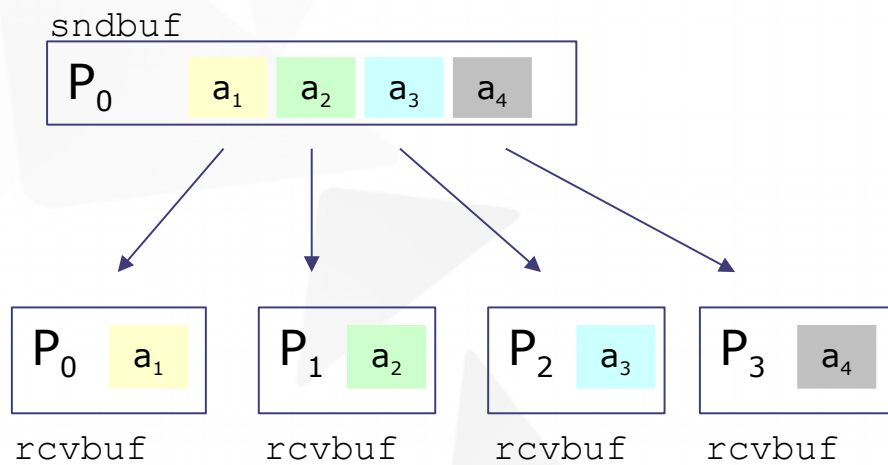
Arguments definition are like other MPI subroutine

rcvcount is the number of elements collected from each process, not the size of **rcvbuf**, that should be **rcvcount** times the number of process in the communicator

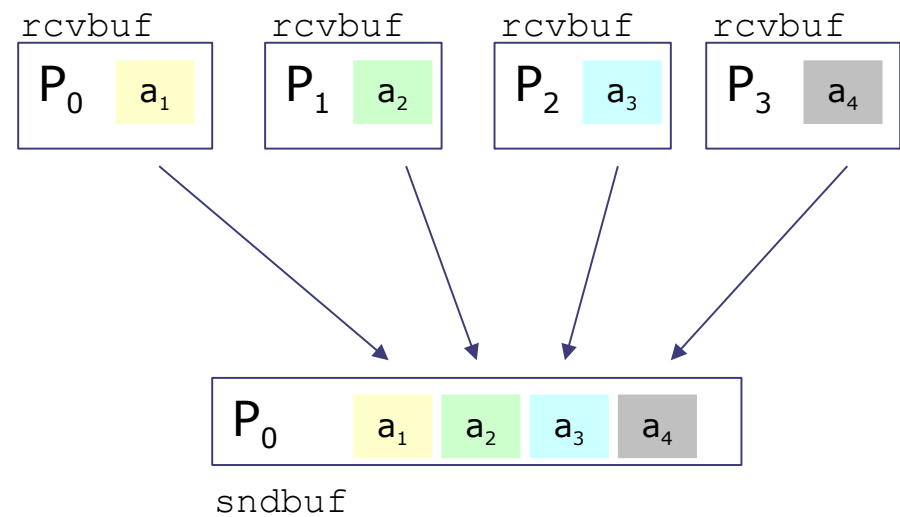
The receiver arguments are significant only at root

Scatter/Gather

Scatter



Gather



Scatter/Gather examples

scatter

```
PROGRAM scatter
INCLUDE 'mpif.h'
INTEGER ierr, myid, nproc, nsnd, I, root
INTEGER status(MPI_STATUS_SIZE)
REAL A(16), B(2)
CALL MPI_INIT(ierr)
CALL MPI_COMM_SIZE(MPI_COMM_WORLD, nproc, ierr)
CALL MPI_COMM_RANK(MPI_COMM_WORLD, myid, ierr)
root = 0
IF( myid .eq. root ) THEN
  DO i = 1, 16
    a(i) = REAL(i)
  END DO
END IF
nsnd = 2
CALL MPI_SCATTER(a, nsnd, MPI_REAL, b, nsnd,
& MPI_REAL, root, MPI_COMM_WORLD, ierr)
WRITE(6,*) myid, ': b(1)=', b(1), 'b(2)=', b(2)
CALL MPI_FINALIZE(ierr)
END
```

gather

```
PROGRAM gather
INCLUDE 'mpif.h'
INTEGER ierr, myid, nproc, nsnd, I, root
INTEGER status(MPI_STATUS_SIZE)
REAL A(16), B(2)
CALL MPI_INIT(ierr)
CALL MPI_COMM_SIZE(MPI_COMM_WORLD, nproc, ierr)
CALL MPI_COMM_RANK(MPI_COMM_WORLD, myid, ierr)
root = 0
b(1) = REAL( myid )
b(2) = REAL( myid )
nsnd = 2
CALL MPI_GATHER(b, nsnd, MPI_REAL, a, nsnd,
& MPI_REAL, root, MPI_COMM_WORLD, ierr)
IF( myid .eq. root ) THEN
  DO i = 1, (nsnd*nproc)
    WRITE(6,*) myid, ': a(i)=', a(i)
  END DO
END IF
CALL MPI_FINALIZE(ierr)
END
```

Exercise:

- Modify pi_mpi.c to use collective operation instead of naive communication algorithm