

General concepts

- ▶ Collective communications allow making a series of point-to-point communications in one single call.
- ▶ A collective communication always concerns all the processes of the indicated communicator.
- ▶ For each process, the call ends when its participation in the collective call is completed, in the sense of point-to-point communications (therefore, when the concerned memory area can be changed).
- ▶ The management of tags in these communications is transparent and system-dependent. Therefore, they are never explicitly defined during calls to subroutines. An advantage of this is that collective communications never interfere with point-to-point communications.

Types of collective communications

There are three types of subroutines :

1. One which ensures global synchronizations: `MPI_Barrier()`
2. Ones which only transfer data :
 - ▶ Global distribution of data : `MPI_Bcast()`,
 - ▶ Selective distribution of data: `MPI_Scatter()`,
 - ▶ Collection of distributed data : `MPI_Gather()`,
 - ▶ Collection of distributed data by all the processes : `MPI_Allgather()`,
 - ▶ Collection and selective distribution by all the processes of distributed data : `MPI_Alltoall()`
3. Ones which, in addition to the communications management, carry out operations on the transferred data:
 - ▶ Reduction operations (sum, product, maximum, minimum, etc.), whether of a predefined or personal type : `MPI_Reduce ()`
 - ▶ Reduction operations with distributing of the result (this is in fact equivalent to an `MPI_Reduce ()` followed by an `MPI_Bcast ()`) : `MPI_Allreduce ()`

Global synchronization : MPI_Barrier ()

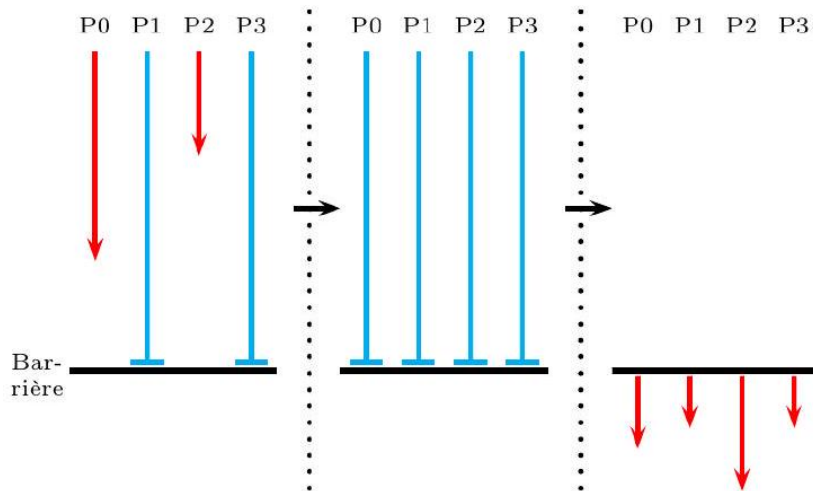


Figure - Global Synchronization : MPI_Barrier()

```
1 subroutine MPI_BARRIER (comm, code)
2   TYPE (MPI_Comm), intent (in)
3   integer, optional, intent(out) :: code
4 end subroutine MPI_BARRIER
```

Global distribution : MPI_Bcast()

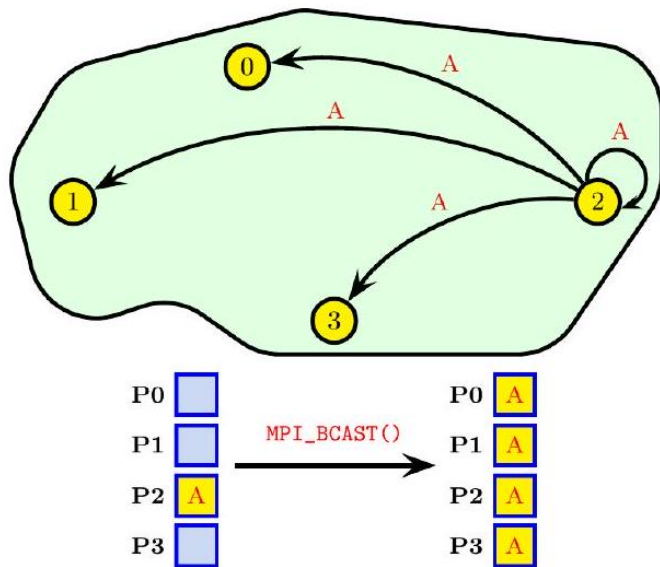


Figure - Global distribution : MPI_Bcast()

Global distribution : MPI_Bcast()

```
1 MPI_BCAST(buffer, count, datatype, root, comm, code)
2 TYPE(),dimension(..) :: buffer
3 integer, intent(in) :: count, root
4 TYPE(MPI_Datatype), intent(in) :: datatype
5 TYPE (MPI_Comm), intent(in) :: comm
6 integer, optional, intent(out) :: code
```

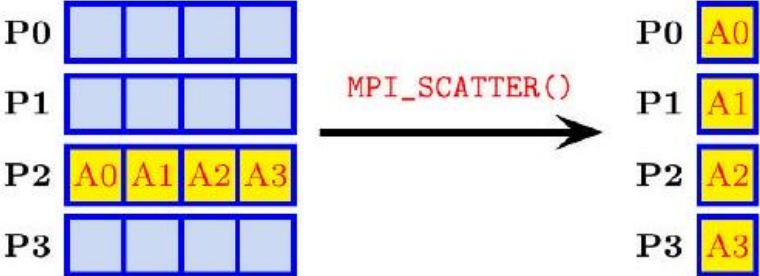
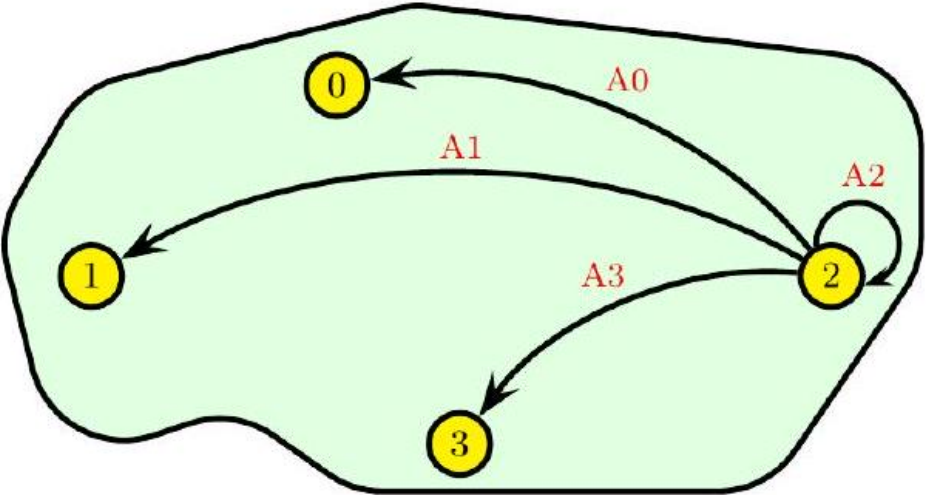
1. Send, starting at position buffer, a message of count element of type datatype, by the root process, to all the members of communicator comm.
2. Receive this message at position buffer for all the processes other than the root.

Example of MPI_Bcast()

```
1  program bcast
2      use mpi_f08
3      implicit none
4      integer :: rank,value
5      call MPI_INIT()
6      call MPI_COMM_RANK(MPI_COMM_WORLD,rank)
7      if (rank == 2) value=rank+1000
8      call MPI_BCAST(value,1,MPI_INTEGER,2,MPI_COMM_WORLD)
9      print *, 'I, process ',rank,', received ',value,' of process 2'
10     call MPI_FINALIZE()
11 end program bcast
```

```
1  > mpiexec -n 4 bcast
2  I, process 2, received 1002 of process 2
3  I, process 0, received 1002 of process 2
4  I, process 1, received 1002 of process 2
5  I, process 3, received 1002 of process 2
```

Selective distribution : MPI_Scatter()



Selective distribution : MPI_Scatter()

```
1 MPI_SCATTER(sendbuf, sendcount, sendtype,  
2 rcvbuf, rcvcount, rcvtype, root, comm, code)  
3 TYPE(*), dimension(..) :: sendbuf, rcvbuf  
4 integer, intent(in) :: sendcount, rcvcount, root  
5 TYPE(MPI_Datatype), intent(in) :: sendtype, rcvtype  
6 TYPE (MPI_Comm), intent(in) :: comm  
7 integer, optional, intent(out) :: code
```

1. Scatter by process root, starting at position sendbuf, message sendcount element of type sendtype, to all the processes of communicator comm.
 2. Receive this message at position rcvbuf, of rcvcount element of type rcvtype for all processes of communicator comm.
- ▶ The couples (sendcount, sendtype) and (rcvcount, rcvtype) must represent the same quantity of data.
 - ▶ Data are scattered in chunks of same size ; a chunk consists of sendcount elements of type sendtype.
 - ▶ The i-th chunk is sent to the i-th process.

Example of MPI_Scatter() I

```
1      program scatter
2          use mpi_f08
3          implicit none
4          integer, parameter :: nb_values=8
5          integer :: nb_procs,rank,block_length,i,code
6          real, allocatable, dimension(:) :: values,recvdata
7          call MPI_INIT()
8          call MPI_COMM_SIZE (MPI_COMM_WORLD,nb_procs)
9          Call MPI_COMM_RANK (MPI_COMM_WORLD, rank)
10         block_length=nb_values/nb_procs
11         allocate(recvdata(block_length))
12         if (rank == 2) then
13             allocate(values(nb_values))
14             values (:) = /(1000. + i, i=1,nb_values)/)
15             print *, 'I, process ',rank,'send my values array : ', &
16         end if
17         call MPI_SCATTER(values,block_length,MPI_REAL, recvdata,block_length, &
18             MPI_REAL, 2,MPI_COMM_WORLD)
19         print *, 'I, process ', rank, ', received', recvdata(1:block_length),  of process 2
20     call MPI_FINALIZE()
21 end program scatter
```

Example of MPI_Scatter() II

```
1 > mpiexec -n 4 scatter
2 I, process 2 send my values array :
3 1001. 1002. 1003. 1004. 1005. 1006. 1007. 1008.
4 I, process 0, received 1001. 1002. of process 2
5 I, process 1, received 1003. 1004. of process 2
6 I, process 3, received 1007. 1008. of process 2
7 I, process 2, received 1005. 1006. of process 2
```

Collection: MPI_Gather()

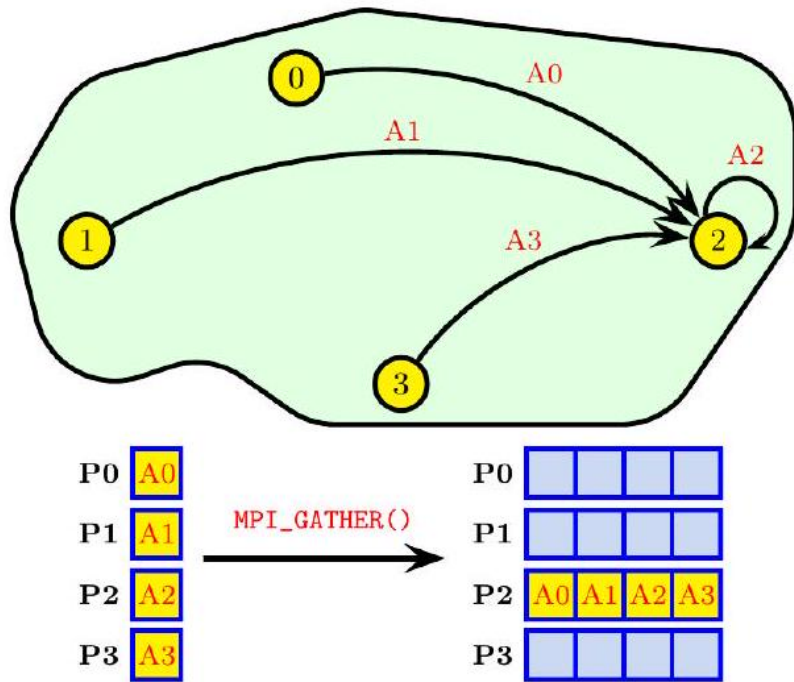


Figure - Collection : MPI_Gather ()

Collection : MPI_Gather()

```
1 MPI_GATHER(sendbuf, sendcount, sendtype,  
2           recvbuf,recvcount, recvtype,root, comm, code)  
3 TYPE(*), dimension(..), intent(in) :: sendbuf  
4 TYPE(*), dimension(..) :: recvbuf  
5 integer, intent(in) :: sendcount, recvcount, root  
6 TYPE (MPI_Datatype), intent(in) :: sendtype, recvtype  
7 TYPE (MPI_Comm), intent(in) :: comm  
8 integer, optional, intent(out) :: code
```

1. Send for each process of communicator comm, a message starting at position sendbuf, of sendcount element type sendtype.
2. Collect all these messages by the root process at position recvbuf, recvcount element of type recvtype.
 - ▶ The couples (sendcount, sendtype) and (recvcount, recvtype) must represent the same size of data.
 - ▶ The data are collected in the order of the process ranks.

Example of MPI_Gather() I

```
1  program gather
2      use mpi_f08
3      implicit none
4      integer, parameter :: nb_values=8
5      integer :: nb_procs,rank,block_length,i
6      real, dimension(nb_values) :: recvdata
7      real, allocatable, dimension(:) :: values
8      call MPI_INIT()
9      call MPI_COMM_SIZE(MPI_COMM_WORLD,nb_procs)
10     call MPI_COMM_RANK(MPI_COMM_WORLD,rank)
11     block_length=nb_values/nb_procs
12     allocate(values(block_length))
13     values(:)=(/(1000.+rank*block_length+i,i=1,block_length)/)
14     print *, ' I, process ',rank,' sent my values array : ',&
15             values(1:block_length)
16     call MPI_GATHER(values,block_length,MPI_REAL,recvdata,block_length, &
17                   MPI_REAL,2,MPI_COMM_WORLD)
18     if (rank == 2) print *, 'I, process 2', ' received ',recvdata(1:nb_values)
19     call MPI_FINALIZE()
20 end program gather
```

Example of MPI_Gather() II

```
1 > mpiexec -n 4 gather
2 I, process 1 sent my values array :1003. 1004.
3 I, process 0 sent my values array :1001. 1002.
4 I, process 2 sent my values array :1005. 1006.
5 I, process 3 sent my values array :1007. 1008.
6 I, process 2, received 1001. 1002. 1003. 1004. 1005. 1006. 1007. 1008.
```

Gather-to-all : MPI_Allgather ()

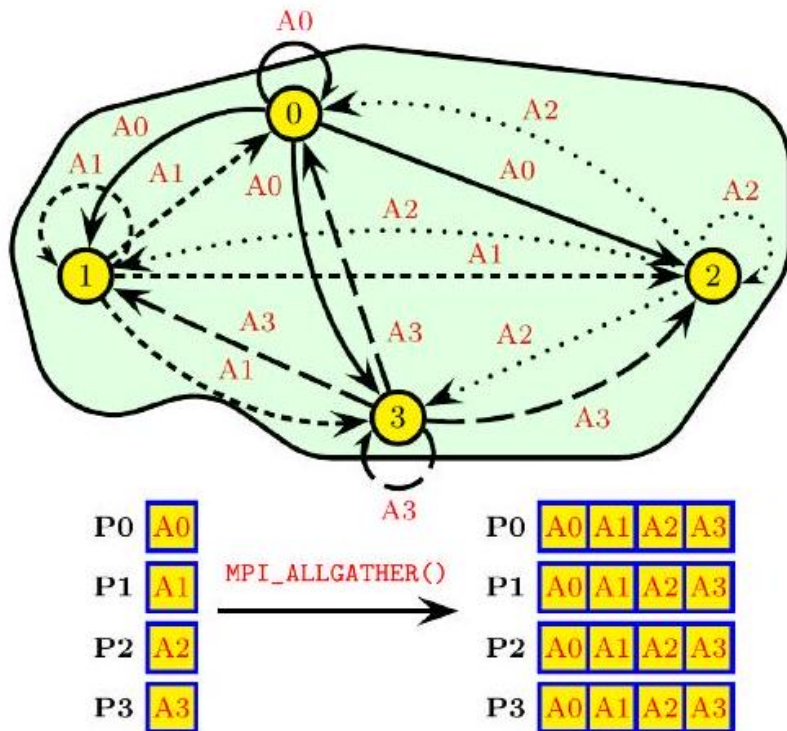


Figure - Gather-to-all : MPI_Allgather()

Gather-to-all : MPI_Allgather ()

Corresponds to an MPI_Gather () followed by an MPI_Bcast () :

```
MPI_ALLGATHER (sendbuf, sendcount, sendtype,  
               recvbuf, recvcount, recvtype, comm, code)
```

```
TYPE(*), dimension(..), intent(in) :: sendbuf
```

```
TYPE(*), dimension(..) :: recvbuf
```

```
integer, intent(in) :: sendcount, recvcount
```

```
TYPE (MPI_Datatype), intent(in) :: sendtype, recvtype
```

```
TYPE (MPI_Comm), intent(in) :: comm
```

```
integer, optional, intent(out) :: code
```

1. Send by each process of communicator comm, a message starting at position sendbuf, of sendcount element, type sendtype.
2. Collect all these messages, by all the processes, at position recvbuf of recvcount element type recvtype.
 - ▶ The couples (sendcount, sendtype) and (recvcount, recvtype) must represent the same data size.
 - ▶ The data are gathered in the order of the process ranks.

Example of MPI_Allgather ()

```
program allgather
  use mpi_f08
  implicit none
  integer, parameter :: nb_values=8
  integer :: nb_procs,rank,block_length,i
  real, dimension(nb_values) :: recvdata
  real, allocatable, dimension(:) :: values
  call MPI_INIT()
  call MPI_COMM_SIZE(MPI_COMM_WORLD,nb_procs)
  call MPI_COMM_RANK(MPI_COMM_WORLD,rank)
  block_length=nb_values/nb_procs
  allocate(values(block_length))
  values(:)=(/(1000.+rank*block_length+i,i=1,block_length)/)
  call MPI_ALLGATHER(values,block_length,MPI_REAL,recvdata,block_length, &
    MPI_REAL,MPI_COMM_WORLD)
  print *, 'I, process ',rank,', received', recvdata(1:nb_values)
  call MPI_FINALIZE()
end program allgather
```

```
mpiexec -n 4 allgather
```

```
I, process 1, received 1001. 1002. 1003. 1004. 1005. 1006. 1007. 1008.
```

```
I, process 3, received 1001. 1002. 1003. 1004. 1005. 1006. 1007. 1008.
```

```
I, process 2, received 1001. 1002. 1003. 1004. 1005. 1006. 1007. 1008.
```

```
I, process 0, received 1001. 1002. 1003. 1004. 1005. 1006. 1007. 1008.
```

Extended gather : MPI_Gatherv ()

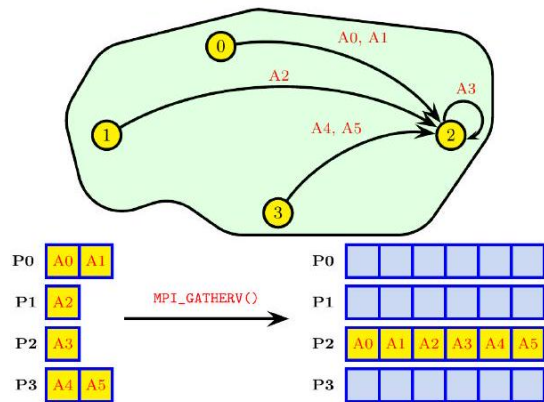


Figure 17 - Extended gather : MPI_Gatherv ()

Extended Gather: MPI_Gatherv ()

This is an MPI_Gather () where the size of messages can be different among processes:

```
MPI_GATHERV (sendbuf, sendcount, sendtype,  
             recvbuf, recvcunts, displs, recvtype,  
             root, comm, code)
```

```
TYPE(*), dimension(..), intent(in) :: sendbuf  
TYPE (*), dimension(..) :: recvbuf  
integer, intent(in) :: sendcount, root  
TYPE (MPI_Datatype), intent(in) :: sendtype, recvtype  
integer, dimension(:), intent(in) :: recvcunts, displs  
TYPE (MPI_Comm), intent(in) :: comm  
integer, optional, intent(out) :: code
```

The i-th process of the communicator comm sends to process root, a message starting at position sendbuf, of sendcount element of type sendtype, and receives at position recvbuf, of recvcunts(i) element of type recvtype, with a displacement of displs(i).

- ▶ The couples (sendcount,sendtype) of the i-th process and (recvcunts(i), recvtype) of process root must be such that the data size sent and received is the same.

Example of MPI_Gatherv ()

```
program gatherv
  use mpi_f08
  implicit none
  INTEGER, PARAMETER :: nb_values=10
  INTEGER :: nb_procs, rank, block_length, i
  REAL, DIMENSION(nb_values) :: recvdata,remainder
  REAL, ALLOCATABLE, DIMENSION(:) :: valueS
  INTEGER, ALLOCATABLE, DIMENSION(:) :: nb_elements_received,displacement
  CALL MPI_INIT()
  CALL MPI_COMM_SIZE (MPI_COMM_WORLD,nb_procs)
  CALL MPI_COMM_RANK(MPI_COMM_WORLD,rank)
  block_length=nb_values/nb_procs
  remainder=mod(nb_values,nb_procs)
  if(rank < remainder) block_length = block_length + 1
  ALLOCATE (values(block_length))
  values(:) = (/ (1000.+(rank*(nb_values/nb_procs))+min(rank,remainder) +i, &
    i=1,block_length})/
  PRINT *, 'I, process ', rank,'sent my values array : ',&
    values(1:block_length)
  IF (rank == 2) THEN
    ALLOCATE (nb_elements_received(nb_procs), displacement (nb_procs))
    nb_elements_received(1) = nb_values/nb_procs
    if (remainder > 0) nb_elements_received(1)=nb_elements_received(1)+1
```

Example of MPI_Gatherv ()

```
CALL MPI_GATHERV(values,block_length,MPI_REAL, recvdata,nb_elements_received, &
    displacement,MPI_REAL,2,MPI_COMM_WORLD)
IF (rank == 2) PRINT *, 'I, process 2, received ', recvdata (1:nb_values)
CALL MPI_FINALIZE()
end program gatherv
```

```
> mpiexec -n 4 gatherv
I, process 0 sent my values array: 1001. 1002. 1003.
I, process 2 sent my values array : 1007. 1008.
I, process 3 sent my values array : 1009. 1010.
I, process 1 sent my values array : 1004. 1005. 1006.
I, process 2 receives 1001. 1002. 1003. 1004. 1005. 1006. 1007. 1008. 1009. 1010.
```

Collection and distribution : MPI_Alltoall ()

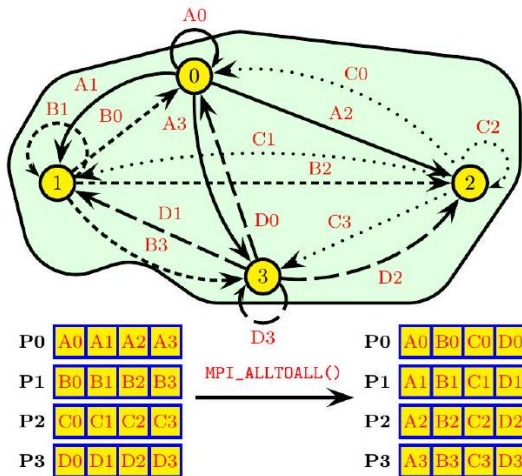


Figure 18 - Collection and distribution : MPI_Alltoall ()

Collection and distribution : MPI_Alltoall ()

```
MPI_ALLTOALL (sendbuf, sendcount, sendtype,  
              recvbuf,recvcount, recvtype, comm, code)  
TYPE(*), dimension(..), intent(in) :: sendbuf  
TYPE (*), dimension(..) :: recvbuf  
integer, intent(in) :: sendcount, recvcount  
TYPE (MPI_Datatype), intent(in) :: sendtype, recvtype  
TYPE(MPI_Comm), intent(in) :: comm  
integer, optional, intent(out) :: code
```

Here, the i-th process sends its j-th chunk to the j-th process which places it in its i-th chunk.

- The couples (sendcount, sendtype) and (recvcount, recvtype) must be such that they represent equal data sizes.

Example of MPI_Alltoall ()

```
program alltoall
  use mpi_f08
  implicit none
  integer, parameter :: nb_values=8
  integer :: nb_procs,rank,block_length,i
  real, dimension(nb_values) :: values,recvdata
  call MPI_INIT()
  call MPI_COMM_SIZE(MPI_COMM_WORLD,nb_procs)
  call MPI_COMM_RANK(MPI_COMM_WORLD,rank)
  values(:)=(/(1000.+rank*nb_values+i,i=1,nb_values)/)
  block_length=nb_values/nb_procs
  print *, 'I, process ',rank,'sent my values array : ',&
    values(1:nb_values)
  call MPI_ALLTOALL(values,block_length,MPI_REAL, recvdata,block_length, &
    MPI_REAL,MPI_COMM_WORLD)
  print *, 'I, process ',rank,', received', recvdata(1:nb_values)
  call MPI_FINALIZE()
```

```
end program alltoall
```

```
> mpiexec -n 4 alltoall
```

```
I, process 1 sent my values array :
```

```
1009. 1010. 1011. 1012. 1013. 1014. 1015. 1016.
```

```
I, process 0 sent my values array :
```

```
1001. 1002. 1003. 1004. 1005. 1006. 1007. 1008.
```


Global reduction

- ▶ A reduction is an operation applied to a set of elements in order to obtain one single value. Typical examples are the sum of the elements of a vector (`SUM (A (:))`) or the search for the maximum value element in a vector (`MAX (V(:))`).
- ▶ MPI proposes high-level subroutines in order to operate reductions on data distributed on a group of processes. The result is obtained on only one process (`MPI_Reduce ()`) or on all the processes (`MPI_Allreduce ()`, which is in fact equivalent to an `MPI_Reduce ()` followed by an `MPI_Bcast ()`).
- ▶ If several elements are implied by process, the reduction function is applied to each one of them (for instance to each element of a vector).

Distributed reduction : MPI_Reduce

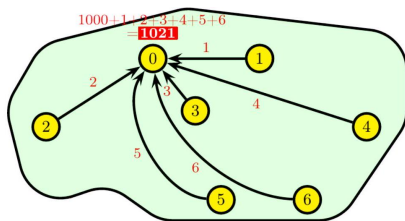


Figure 19 - Distributed reduction (sum)

Operations

Name	Operation
MPI_SUM	Sum of elements
MPI_PROD	Product of elements
MPI_MAX	Maximum of elements
MPI_MIN	Minimum of elements
MPI_MAXLOC	Maximum of elements and location
MPI_MINLOC	Minimum of elements and location
MPI_LAND	Logical AND
MPI_LOR	Logical OR
MPI_LXOR	Logical exclusive OR

Global reduction : MPI_Reduce ()

```
MPI_REDUCE (sendbuf, recvbuf, count, datatype, op, root, comm, code)
TYPE(*), dimension(..), intent(in ) :: sendbuf
TYPE(*), dimension(..) :: recvbuf
integer, intent(in) :: count, root
TYPE (MPI_Datatype), intent(in) :: datatype
TYPE (MPI_Op), intent(in) :: op
TYPE(MPI_Comm), intent(in) :: comm
integer, optional, intent(out) :: code
```

1. Distributed reduction of count elements of type datatype, starting at position sendbuf, with the operation op from each process of the communicator comm,
2. Return the result at position recvbuf in the process root.