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 ()

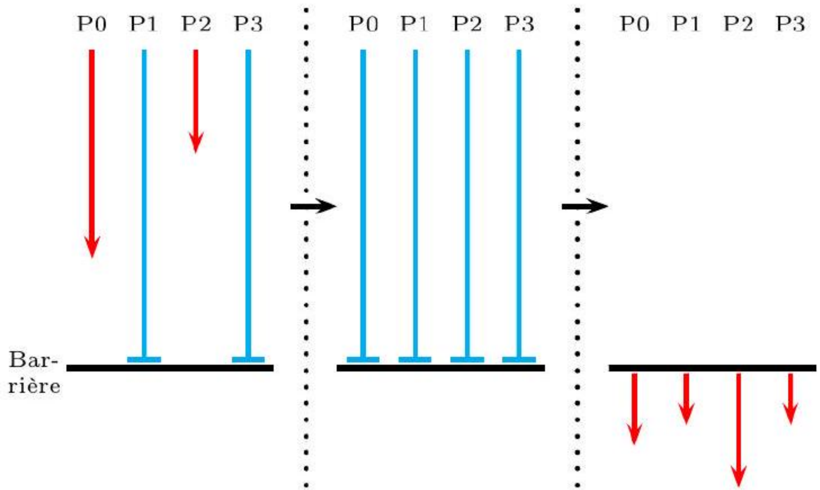


Figure - Global Synchronization : MPI_Barrier()

```fortran
subroutine MPI_BARRIER (comm, code)
    TYPE (MPI_Comm), intent (in)
    integer, optional, intent(out) :: code
end subroutine MPI_BARRIER
```

Global distribution : MPI_Bcast()

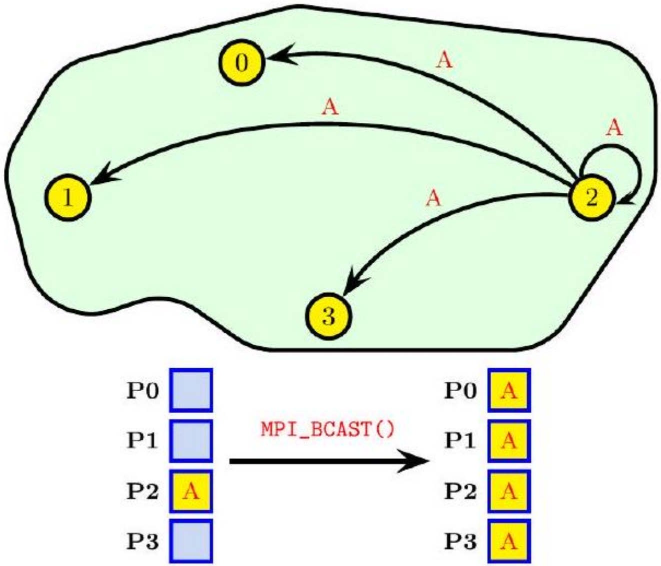


Figure - Global distribution : MPI_Bcast()

Global distribution : MPI_Bcast()

```
MPI_BCAST(buffer, count, datatype, root, comm, code)
TYPE(),dimension(..) :: buffer
integer, intent(in) :: count, root
TYPE(MPI_Datatype), intent(in) :: datatype
TYPE (MPI_Comm), intent(in) :: comm
integer, optional, intent(out) :: code
```

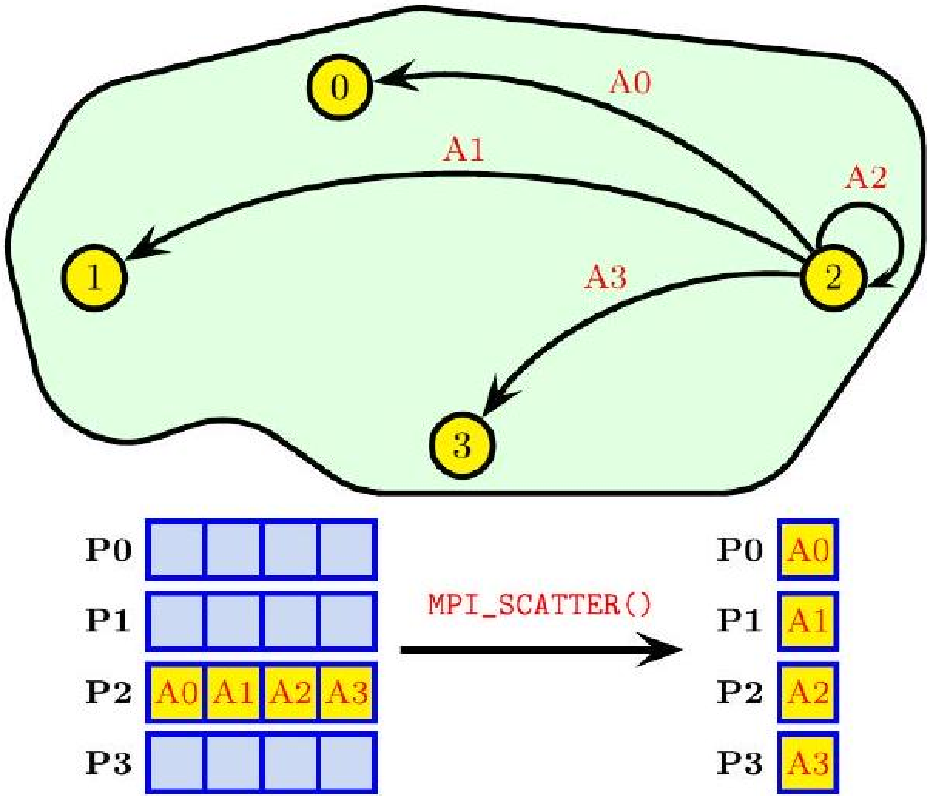
1. Send, starting at position buffer, a message of count element of type datatype, by the root process, to all the members of communicator comm.
2. Receive this message at position buffer for all the processes other than the root.

Example of MPI_Bcast()

```fortran
program bcast
    use mpi_f08
    implicit none
    integer :: rank,value
    call MPI_INIT()
    call MPI_COMM_RANK(MPI_COMM_WORLD,rank)
    if (rank == 2) value=rank+1000
    call MPI_BCAST(value,1,MPI_INTEGER,2,MPI_COMM_WORLD)
    print *,'I, process ',rank,', received ',value,' of process 2'
    call MPI_FINALIZE()
end program bcast
```

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

Selective distribution : `MPI_Scatter()`



Selective distribution : MPI_Scatter()

```
MPI_SCATTER(sendbuf, sendcount, sendtype,
recvbuf, recvcount, recvtype, root, comm, code)
TYPE(*), dimension(..) :: sendbuf, recvbuf
integer, intent(in) :: sendcount, recvcount, root
TYPE(MPI_Datatype), intent(in) :: sendtype, recvtype
TYPE (MPI_Comm), intent(in) :: comm
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 recvbuf, of recvcount element of type recvtype for all processes of communicator comm.

- The couples (sendcount, sendtype) and (recvcount, recvtype) 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

```
        program scatter
            use mpi_f08
            implicit none
            integer, parameter :: nb_values=8
            integer :: nb_procs,rank,block_length,i,code
            real, allocatable, dimension(:) :: values,recvdata
            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(recvdata(block_length))
            if (rank == 2) then
                allocate(values(nb_values))
                values (:) = (/(1000. + i, i=1,nb_values)/)
                print *,'I, process ',rank,'send my values array : ', &
            end if
            call MPI_SCATTER(values,block_length,MPI_REAL, recvdata,block_length, &
            MPI_REAL, 2,MPI_COMM_WORLD)
           print *,',I, process ', rank,', received', recvdata(1:block_length),  of process 2
    call MPI_FINALIZE()
    end program scatter
```

Example of MPI_Scatter() II

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

Collection: MPI_Gather()

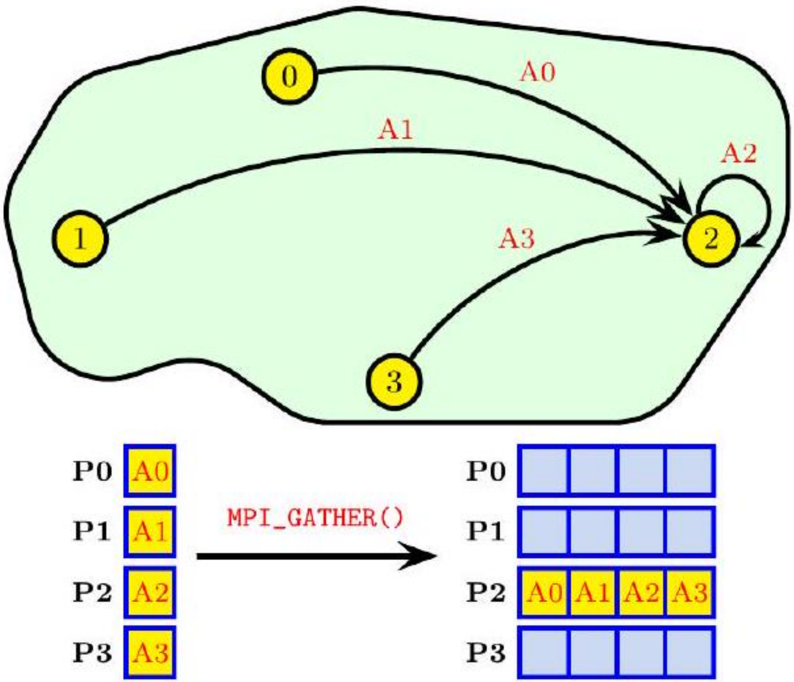


Figure - Collection : MPI_Gather ()

Collection : MPI_Gather()

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

```fortran
program gather
    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)/)
    print *,' I, process ',rank,'sent my values array : ',&
            values(1:block_length)
    call MPI_GATHER(values,block_length,MPI_REAL,recvdata,block_length, &
            MPI_REAL,2,MPI_COMM_WORLD)
    if (rank == 2) print *,'I, process 2', ' received ',recvdata(1:nb_values)
    call MPI_FINALIZE()
end program gather
```

Example of MPI_Gather() II

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

Gather-to-all : MPI_Allgather ()

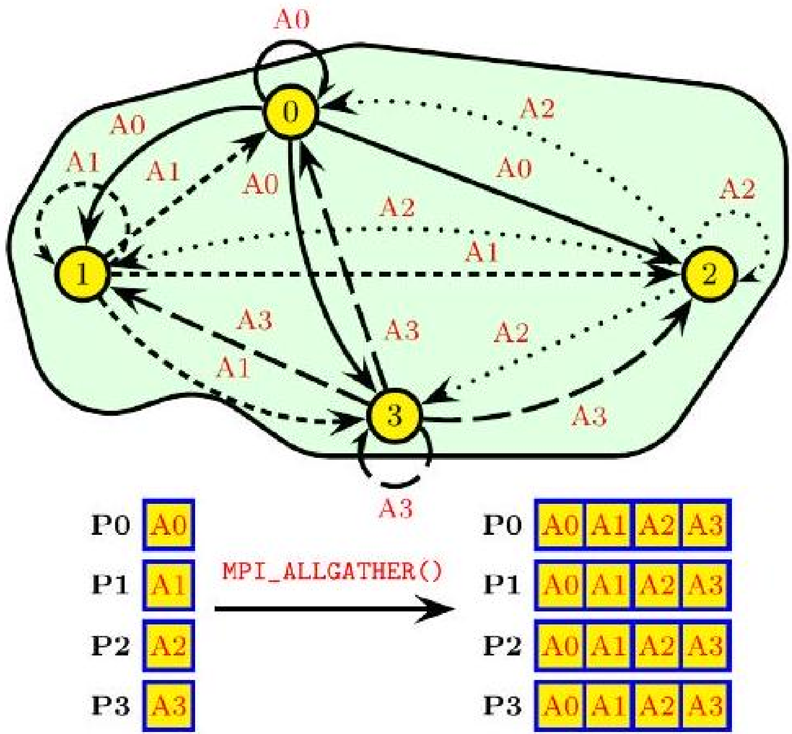


Figure - Gather-to-all : MPI_Allgather()

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 ()

```fortran

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 ()

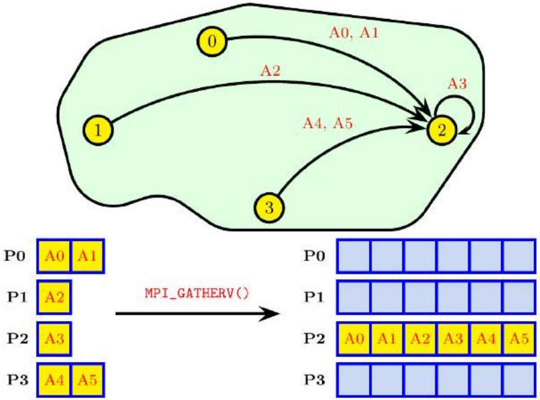


Figure - Extended gather : `MPI_Gatherv()`

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, recvcounts, displs, recvtype,
    root, comm, code)
```

```fortran
TYPE(*), dimension(..), intent(in) :: sendbuf
TYPE (*), dimension(..) :: recvbuf
integer, intent(in) :: sendcount, root
TYPE (MPI_Datatype), intent(in) :: sendtype, recvtype
integer, dimension(:), intent(in) :: recvcounts, 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 recvcounts(i) element of type recvtype, with a displacement of displs(i).

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

Example of MPI_Gatherv ()

```fortran
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
        displacement (1) = 0
        DO i=2,nb_procs
            displacement(i) = displacement(i-1)+nb_elements_received(i-1)
            nb_elements_received(i) = nb_values/nb_procs
            if (i-1 < remainder) nb_elements_received(i)=nb_elements_received(i)+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 ()

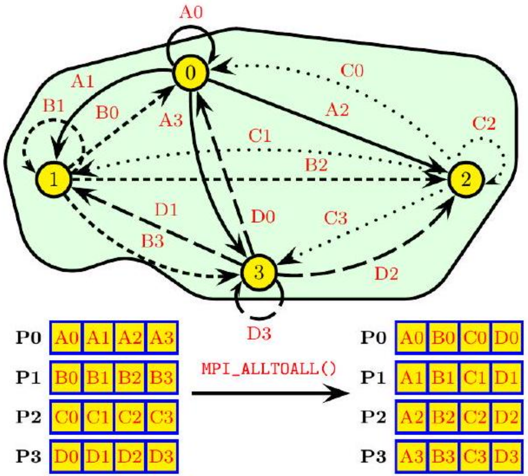


Figure 18 - Collection and distribution : MPI_Alltoall ()

Collection and distribution : MPI_Alltoall ()

```fortran
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 ()

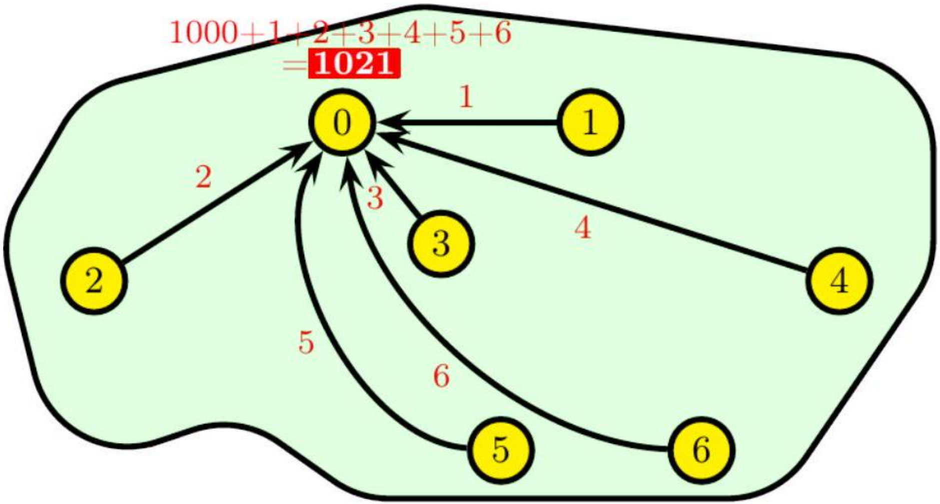
```fortran
program alltoall
    use mpi_f08
    implicit none
    integer, parameter :: nb_values=8
    integer :: nb_procs,rank,block_length,i
    real, dimension(nb_values) :: values,recvdata
    call MPI_INIT()
    call MPI_COMM_SIZE(MPI_COMM_WORLD,nb_procs)
    call MPI_COMM_RANK(MPI_COMM_WORLD,rank)
    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.
I, process 2 sent my values array :
1017. 1018. 1019. 1020. 1021. 1022. 1023. 1024.
I, process 3 sent my values array :
1025. 1026. 1027. 1028. 1029. 1030. 1031. 1032.
```

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



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.

Example of MPI_Reduce()

```
program reduce
    use mpi_f08
    implicit none
    integer :: nb_procs,rank,value,sum
    call MPI_INIT()
    call MPI_COMM_SIZE(MPI_COMM_WORLD,nb_procs)
    call MPI_COMM_RANK(MPI_COMM_WORLD,rank)
    if (rank == 0) then
        value=1000
    else
        value=rank
    endif
    call MPI_REDUCE(value,somme,1,MPI_INTEGER,MPI_SUM,0,MPI_COMM_WORLD)
    if (rank == 0) then
        print *,'I, process 0, have the global sum value ',sum
    end if
    call MPI_FINALIZE()
end program reduce
```

```
$>$ mpiexec - n 7 reduce\\
I, process 0, have the global sum value 1021
```

Distributed reduction with distribution of the result : `MPI_Allreduce()`

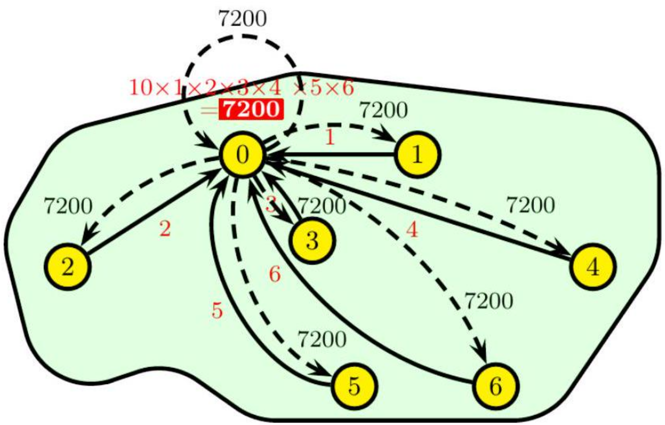


Figure - Distributed reduction (product) with distribution of the result

Global all-reduction : `MPI_Allreduce()`

```
MPI_ALIREDUCE (sendbuf,recvbuf,count, datatype,op,comm, code)
TYPE(*), dimension(..), intent(in) :: sendbuf
TYPE(*), dimension(..) :: recvbuf
integer, intent(in) :: count
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. Write the result at position recvbuf for all the processes of the communicator comm.

Example of MPI_Allreduce ()

```
program allreduce
    use mpi_f08
    implicit none
    integer :: nb_procs,rank,value,product
    call MPI_INIT()
    call MPI_COMM_SIZE(MPI_COMM_WORLD,nb_procs)
    call MPI_COMM_RANK(MPI_COMM_WORLD,rank)
    if (rank == 0) then
        value=10
    else
        value=rank
    endif
    call MPI_ALLREDUCE(value,product,1,MPI_INTEGER,MPI_PROD,MPI_COMM_WORLD)
    print *,'I,process ',rank,', received the value of the global product ', product
    call MPI_FINALIZE()
end program allreduce
```

Example of MPI_Allreduce ()

```
> mpiexec -n }7\mathrm{ allreduce
I, process 6, received the value of the global product }720
I, process 2, received the value of the global product }720
I, process 0, received the value of the global product }720
I, process 4, received the value of the global product }720
I, process 5, received the value of the global product }720
I, process 3, received the value of the global product }720
I, process 1, received the value of the global product }720
```

Additions

- The MPI_Scan () subroutine allows making partial reductions by considering, for each process, the previous processes of the communicator and itself. MPI_Exscan () is the exclusive version of MPI_Scan (), which is inclusive.
- The MPI_Op_create () and MPI_Op_free () subroutines allow personal reduction operations.
- For each reduction operation, the keyword MPI_IN_PLACE can be used in order to keep the result in the same place as the sending buffer (but only for the rank(s) that will receive results). Example : call MPI_Allreduce(MPI_IN_PLACE,sendrecvbuf, . . .).

Additions

- Similarly to what we have seen for MPI_Gatherv () with repect to MPI_Gather(), the MPI_Scatterv(), MPI_Allgatherv() and MPI_Alltoallv() subroutines extend MPI_Scatter(), MPI_Allgather() and MPI_Alltoall () to the cases where the processes have different numbers of elements to transmit or gather.
- MPI_Alltoallw () is the version of MPI_Alltoallv () which enables to deal with heterogeneous elements (by expressing the displacements in bytes and not in elements).

MPI Hands-On - Exercise 3 : Collective communications and reductions

- The aim of this exercice is to compute pi by numerical integration. $\pi = \int_0^1 \frac{4}{1+x^2} dx$.
- We use the rectangle method (mean point).
- Let $f(x) = \frac{4}{1+x^2}$ be the function to integrate.
- nbblock is the number of points of discretization.
- width $= \frac{1}{\text{nbblock}}$ the length of discretization and the width of all rectangles.
- Sequential version is available in the pi.f90 source file.
- You have to do the parallel version with MPI in this file.