Chapter 1

Coarrays in XcalableMP

1.1 Purposes

The coarray features in Fortran 2008 are extended and integrated into XcalableMP. The specifications in this chapter are designed to achieve the following purposes:

- Upward compatibility to the Fortran 2008 coarray features
  If an XcalableMP program does not contain any XMP directives, any standard-conforming Fortran 2008 program remains standard-conforming under XcalableMP. In this sense, the interpretations and extensions defined in this chapter are upward compatible with the Fortran International Standard, ISO/IEC 1539-1:2010 (Fortran 2008).

- Support for task parallelism
  XcalableMP makes it possible to construct a task parallel program by combining multiple Fortran 2008 codes, which might be developed independently, with minimum modifications.

- Integration of global-view style programming and local-view style programming
  In XcalableMP, users can use global-view style programming of XcalableMP or local-view style programming, which is typically used in MPI or Fortran 2008 programs, appropriately according to the characteristics of code in a program.

- Possibility of the support for multiple topologies of a computing system
  An XcalableMP processor may allow users to specify the correspondence between node arrays and the topologies of a computing system and exploit the full potential of a particular system.

1.2 Terms and definitions

An image is an instance of an XcalableMP program. A node array is a set of nodes, whose individual elements are arranged in a rectangular pattern. Each node or each element of an instance of a node array uniquely corresponds to an image. This section defines how the image index of an image in a set of images is determined in association with a node array and a TASK directive construct.

1.2.1 Primary image index

Every image has a default image index in all the images at the invocation of a program. In XcalableMP, the default image index is the primary image index and is an integer value in the range one to the number of images at the invocation of a program.

1.2.2 Primary node array

A primary node array is a node array declared with "=**" in a NODES directive. A primary node array corresponds to all the images at the invocation of a program, and also corresponds to all the nodes at the invocation of a program.

The primary image index of an image is the (Fortran) subscript order value of the uniquely corresponding element of a primary node array. The primary node number of a node is also the (Fortran) subscript order value of the uniquely corresponding element of a primary node array.

1.2.3 Image index determined by a *task-directive*

Execution of a *task-directive* determines that a set of nodes (and the corresponding set of images) forms an executing node set. If a name of a node array or a subobject of a node array appears in the *task-directive*, the nodes and the corresponding images in the executing node set are ordered in (Fortran) array element order in the node array or the subobject of the node array. If a name of a template array or a subobject of a template array appears in the *task-directive*, the nodes and the corresponding images in the executing node set are ordered in (Fortran) array element order in the corresponding subobject of the node array. The image index of an image in the determined set of images is the integer order value in the range one to the cardinality of the set of images. The node number of a node in the determined set of nodes is the integer order value in the range one to the cardinality of the set of nodes.

1.2.4 Current image index

The current set of images is a set of images determined by the most lately executed *task-directive* in the TASK directive constructs that are not completed if any TASK directive constructs are being executed. The image index of an image in the current set of images is the current image index.

The current set of images corresponds to primary node arrays and all the nodes at the invocation of a program if there are no TASK directive constructs that are not completed. In this case, the current image index of an image is the same as the primary image index.

1.2.5 Non-primary node array

A non-primary node array is a node array declared without "=*node-ref*", "=**", or "=*" in a NODES directive. A non-primary node array corresponds to all the images at the invocation of a program, and also corresponds to all the nodes at the invocation of a program.

The correspondence between each image and each element of a non-primary node array is processor-dependent. A processor may support any means to specify the correspondence.

The image index of an image in all the images at the invocation of a program is the subscript order value of the corresponding element of a non-primary node array if and only if the current set of images corresponds to the non-primary node whole array in which the nodes in the executing node set are ordered in (Fortran) array element order in the non-primary node whole array. The image index is a non-primary image index.

The correspondence between the primary image index and a non-primary image index of the same image is processor-dependent. Between any two distinct non-primary node arrays, the correspondence between a non-primary image index and another non-primary image index of the same image is processor-dependent unless they have the same shape. If two non-primary node arrays have the same shape, the corresponding elements of the node arrays correspond to the same image.

1.2.6 Executing node array

An executing node array is a node array declared with "=*" in a NODES directive. An executing node array corresponds to the executing node set and also the current set of images at the evaluation of the declaration of the node array.

Each image in the current set of images corresponds to the element of an executing node array whose subscript order value is the same as the current image index of the image at the evaluation of the declaration of the executing node array.

1.2.7 Image index determined by an equivalenced node array

A NODES directive with "=*node-ref*" that is not "=*" or "=**" specifies that each element of the declared node array corresponds in (Fortran) array element order to that of the *node-ref*, which is a name of a node array or a subobject of a node array. The nodes in the declared node array and the corresponding images are ordered in (Fortran) array element order in the *node-ref*. The image index of an image in the set of images corresponding to the declared node array is the integer order value in the range one to the cardinality of the set of images. The node number of a node in the set of nodes corresponding to the declared node array is the integer order value in the range one to the cardinality of the set of nodes.

1.2.8 On-node image index

XcalableMP supports COARRAY directive and IMAGE directive to specify that an image index indicates the image corresponding to the element of a particular node array whose subscript order value is the same as the image index. The image index is an on-node image index for the specified node array. Since evaluation of the declaration of a node array determines a set of images corresponding to the node array, the directives specify that the set of images is the "all images" for the image indices the directives affect. In particular, the on-node image index for a primary node array is the primary image index.

XcalableMP also supports intrinsic procedures to translate image indices between different sets of images.

1.3 Basic concepts

In XcalableMP, "all images" in Fortran 2008 changes coupled with the execution of TASK directive constructs and means the current set of images. In particular, when an ALLOCATE statement is executed for which an *allocate-object* is a coarray, there is an implicit synchronization of all the images in the current set of images. On each image in the current set of images, execution of the segment following the statement is delayed until all other images in the set have executed the same statement the same number of times. When a DEALLOCATE statement is executed for which an *allocate-object* is a coarray, there is an implicit synchronization of all the images in the current set of images. On each image in the current set of images, execution of the segment following the statement is delayed until all other images in the set have executed the same statement the same number of times.

The image index determined by an image selector indicates the current image index by default. Coarrays are visible within the range of the "all images" and accessed with the current image index by default. The image index that appears in an executable statement indicates the current image index by default.

- In the following code fragment, the value of a coarray `b` on the images 1, 2, 3, and 4, which constitute the executing node set and correspond to `node(5)`, `node(6)`, `node(7)`, and `node(8)` respectively, is defined with the value of the coarray `a` on `node(5)`.

```
      program xmpcoarray
```

```
!$xmp nodes node(8)=**   ! A primary node array.
!$xmp task on node(5:8)  ! The executing node set
        call sub         ! corresponds to node(5:8).
!$xmp end task
      end

      subroutine sub
      real, save :: a[*], b[*] ! The images 1, 2, 3,
         :                     ! and 4 correspond to node(5:8),
      b = a[1]                 ! respectively.
```

- In the following code fragment, an allocatable coarray a is allocated on the images 1, 2, 3, and 4, which constitute the executing node set and correspond to node(5), node(6), node(7), and node(8) respectively.

```
      program xmpcoarray
!$xmp nodes node(8)=**
!$xmp task on node(5:8)  ! The executing node set
        call sub2        ! corresponds to node(5:8).
!$xmp end task
      end

      subroutine sub2
      real, allocatable :: a(:)[:]
        :
      allocate(a(0:99)[*])
```

Note

- The result value of xmp_num_nodes() is always the same as that of NUM_IMAGES().
- The result value of xmp_node_num() is always the same as that of THIS_IMAGE().
- In a READ statement, an io-unit that is an asterisk identifies an external unit that is preconnected for sequential formatted input only on the image whose primary image index is 1.

1.3.1 A restriction on allocatable coarrays

When an allocatable coarray is allocated during the execution of TASK directive constructs, the coarray shall be subsequently deallocated before the completion of the TASK directive construct whose *task-directive* is the most lately executed one in the TASK directive constructs that are not completed at the allocation.

1.4 COARRAY directive

1.4.1 Purpose and form of the COARRAY directive

The COARRAY directive maps coarrays onto a node array and the set of images that corresponds to the node array. An image index determined by an image selector for a coarray that appears in a COARRAY directive always indicates the on-node image index for the node array; that is, the specified image corresponds to the node whose subscript order value in the node array is the same as the image index.

A coarray appearing in a COARRAY directive is an on-node coarray of the node array that is specified in the CORRAY directive.

coarray-directive **is** `coarray on` *node-name* `::` *object-name-list*

- An *object-name* shall be a name of a coarray declared in the same scoping unit.

- The same *object-name* shall not appear more than once in COARRAY directives in a scoping unit.

- If an *object-name* is a name of an allocatable object, the current set of images at the allocation and the deallocation of the object shall correspond to the node array specified as the *node-name* and the current image index of each image shall be the same as the subscript order value of the corresponding element of the node array.

- If an *object-name* is a name of an allocated allocatable dummy argument, the set of images onto which it is mapped shall be a subset of the set of images that has allocated most lately the corresponding argument in the chain of argument associations.

- If an *object-name* is a name of a nonallocatable dummy argument whose ultimate argument has allocatable attribute, the set of images onto which the *object-name* is mapped shall be a subset of the set of images that has allocated most lately the corresponding argument in the chain of argument associations.

- The image index determined by an image selector for an on-node coarray shall be in the range of one to the size of the node array onto which the on-node coarray is mapped.

- THIS_IMAGE(COARRAY[,DIM]) shall be invoked by the image contained in the set of images onto which the COARRAY argument is mapped, if the COARRAY argument appears in a COARRAY directive.

Note

- The result value of THIS_IMAGE(COARRAY) is the sequence of cosubscript values for the COARRAY argument that would specify the current image index of the invoking image, if the COARRAY argument does not appear in a COARRAY directive. The result value of THIS_IMAGE(COARRAY) is the sequence of cosubscript values for the COARRAY argument that would specify the on-node image index of the invoking image for the node array onto which the COARRAY argument is mapped, if the COARRAY argument appears in a COARRAY directive.

- The result value of THIS_IMAGE(COARRAY,DIM) is the value of cosubscript DIM in the sequence of cosubscript values for the COARRAY argument that would specify the current image index of the invoking image, if the COARRAY argument does not appear in a COARRAY directive. The result value of THIS_IMAGE(COARRAY,DIM) is the value of cosubscript DIM in the sequence of cosubscript values for the COARRAY argument that would specify the on-node image index of the invoking image for the node array onto which the COARRAY argument is mapped, if the COARRAY argument appears in a COARRAY directive.

1.4.2 An example of the COARRAY directive

```
      module global
!$xmp nodes node(8)=**
      real s[*]             ! The coarray s is always
!$xmp coarray on node :: s  ! visible on node(1:8).
      end global

      program coarray
      use global
!$xmp task on node(5:8)     ! The executing node set
        call sub            ! consists of node(5:8).
!$xmp end task
      end

      subroutine sub
      use global
      real, save :: a[*]    ! The images 1, 2, 3, and 4
         :                  ! correspond to node(5:8), respectively.
      if(this_image().eq.1)then ! The value of the coarray a on node(5)
         s[1] = a               ! defines that of the coarray s on node(1)
      endif
```

1.5 IMAGE directive

1.5.1 Purpose and form of the IMAGE directive

The IMAGE directive specifies that an image index in the following executable statement indicates the on-node image index of the node array specified in the IMAGE directive unless the image index is determined by an image selector.

The IMAGE directive also specifies that execution of a SYNC ALL statement performs a synchronization of all the images corresponding to the node array specified in the IMAGE directive.

image-directive **is** `image (` *node-name* `)`

- An *image-directive* shall be followed by a sync all statement, an image control statement that contains *image-set*, or a reference to an intrinsic procedure that has `IMAGES` argument.

1.5.2 An example of the IMAGE directive

```
      module global
!$xmp nodes node(8)=**
      real s[*]            ! The coarray s is always visible
!$xmp coarray on node :: s  ! on node(1:8).
      end global

      program image
      use global
!$xmp tasks
!$xmp task on node(1:4)
        call subA  ! The executing node set consists of node(1:4).
!$xmp end task
!$xmp task on node(5:8)
        call subB  ! The executing node set consists of node(5:8).
!$xmp end task
!$xmp end tasks
      end

      subroutine subA
      use global
      real, save :: a[*] ! The images 1, 2, 3, and 4
         :               ! correspond to node(1:4), respectively.
!$xmp image(node)        ! Synchronization between node(1:4) and
        sync images(5)   ! node(5).
      a = s[1]           ! a on node(1:4) is defined with
         :               ! the value of s on node(1).
      end subroutine
      subroutine subB
      use global
      real, save :: b[*] ! The images 1, 2, 3, and 4
         :               ! correspond to node(5:8), respectively.
      if(this_image() .eq. 1)then ! The image 1 indicates node(5).
        s[1] = b      ! s on node(1) is defined with the value of
                      ! b on node(5).
!$xmp   image(node)                ! Synchronization between
          sync images((/1,2,3,4/)) ! node(5) and node(1:4).
      endif
         :
      end subroutine
```

1.6 Image index translation intrinsic procedures

1.6.1 Translation to the primary image index

xmp_get_primary_image_index(NUMBER,INDEX,PRI_INDEX,NODE_DESC)

Description. Translate image indices to the primary image indices.

Class. Subroutine.

Arguments. **NUMBER** shall be a scalar of type default integer. It is an INTENT(IN) argument.

INDEX shall be a rank-one array of type default integer. The size of **INDEX** shall be greater than or equal to the value of **NUMBER**. It is an INTENT(IN) argument. The value of each element of **INDEX** shall be in the range one to the size of the node array specified in **NODE_DESC** if **NODE_DESC** appears. The value of each element of **INDEX** shall be in the range one to the cardinality of the current set of images if **NODE_DESC** does not appear.

PRI_INDEX shall be a rank-one array of type default integer. The size of **PRI_INDEX** shall be greater than or equal to the value of **NUMBER**. It is an INTENT(OUT) argument. If **NODE_DESC** appears, **PRI_INDEX(i)** is assigned the primary image index corresponding to the element of the node array specified in **NODE_DESC** whose subscript order value is **INDEX(i)**; otherwise, **PRI_INDEX(i)** is assigned the primary image index corresponding to the image whose current image index is **INDEX(i)**.

NODE_DESC (optional) shall be a descriptor of a node array. It is an INTENT(IN) argument. **NODE_DESC** shall appear in XcalableMP C.

Example. In the following code fragment, the value of `index(1:4)` is `(/5,6,7,8/)`.

```
!$xmp nodes node(1:8)=**        ! A primary node array
!$xmp nodes subnode(4)=node(5:8)
      integer index(4)
      call xmp_get_primary_image_index&
           &(4,(/1,2,3,4/),index,xmp_desc_of(subnode))
```

1.6.2 Translation to the current image index

xmp_get_image_index(NUMBER,INDEX,CUR_INDEX,NODE_DESC)

Description. Translate image indices to the current image indices.

Class. Subroutine.

Arguments. **NUMBER** shall be a scalar of type default integer. It is an INTENT(IN) argument.

INDEX shall be a rank-one array of type default integer. The size of **INDEX** shall be greater than or equal to the value of **NUMBER**. It is an INTENT(IN) argument. The value of each element of **INDEX** shall be in the range one to the size of the node array specified in **NODE_DESC**.

CUR_INDEX shall be a rank-one array of type default integer. The size of **CUR_INDEX** shall be greater than or equal to the value of **NUMBER**. It is an INTENT(OUT) argument. If the current image index corresponding to the element of the node-array specified in **NODE_DESC** whose subscript order value is **INDEX(i)** exists, **CUR_INDEX(i)** is assigned the current image index; otherwise, **CUR_INDEX(i)** is assigned zero.

NODE_DESC shall be a descriptor of a node array. It is an INTENT(IN) argument.

Example. In the following code fragment, the value of `index(1:4)` is `(/1,2,3,4/)`.

```
!$xmp nodes node(1:8)=**
      integer index(4)
!$xmp task on node(5:8)
        call xmp_get_image_index&
             &(4,(/5,6,7,8/),index,xmp_desc_of(node))
!$xmp end task
```

1.7 Examples of communication between tasks

- In the following program fragment, two tasks communicate with each other with synchronization.

```
      module nodes
!$xmp nodes node(8)=**            ! A primary node array
      integer, parameter :: n=2
!$xmp nodes subnodeA(n)=node(1:n)      ! subnodeA is for taskA.
!$xmp nodes subnodeB(8-n)=node(n+1:8)  ! subnodeB is for taskB.
      endmodule

      module intertask
      use nodes
      real,save :: dA[*],dB[*]
      endmodule

      use nodes
!$xmp tasks
!$xmp task on subnodeA  ! The taskA is invoked on subnodeA.
        call taskA
!$xmp end task
!$xmp task on subnodeB  ! The taskB is invoked on subnodeB.
        call taskB
!$xmp end task
!$xmp end tasks
      end

      subroutine taskA
      use intertask
         :
      me = this_image()    ! The value of me is i on subnodeA(i).
      if(me.eq.1)then
        call xmp_get_primary_image_index&  ! The value of iyouabs
               &(1,(/1/),iyouabs,subnodeB) ! is n+1.
!$xmp   image(node)                       ! Synchronization between
          sync images(iyouabs)            ! node(1) and node(n+1).
        call exchange(dA,dB,iyouabs)
!$xmp   image(node)                       ! Synchronization between
          sync images(iyouabs)            ! node(1) and node(n+1).
      endif
      sync all                 ! Synchronization within subnodeA.
      if(me.ne.1)dA = dA[1]
      sync all                 ! Synchronization within subnodeA.
       :
      end

      subroutine taskB
      use intertask
         :
      me = this_image()    ! The value of me is i on subnodeB(i).
      if(me.eq.1)then
        call xmp_get_primary_image_index&  ! The value of iyouabs
             &(1,(/1/),iyouabs,subnodeA)   ! is 1.
!$xmp   image(node)                       ! Synchronization between
          sync images(iyouabs)            ! node(n+1) and node(1).
        call exchange(dB,dA,iyouabs)
!$xmp   image(node)                       ! Synchronization between
          sync images(iyouabs)            ! node(n+1) and node(1).
      endif
      sync all                ! Synchronization within subnodeB.
      if(me.ne.1)dB = dB[1]
      sync all                ! Synchronization within subnodeB.

      end

      subroutine exchange(mine,yours,iput)
      use nodes
      real :: mine[*],yours[*]        ! mine and yours are always
!$xmp coarray on node :: mine,yours  ! visible on node(1:8).

      yours[iput] = mine  ! node(1) puts mine to yours[n+1] and
                          ! node(n+1) puts mine to yours[1].
      end
```

- In the following program fragment, two tasks communicate with each other without one-to-one synchronization.

```
!$xmp nodes node(8)=**        ! A primary node array
        :
!$xmp tasks
!$xmp   task on(node(1:n))
          call taskA(n)       ! The taskA is invoked on node(1:n)
!$xmp   end task
!$xmp   task on(node(n+1:8))
          call taskB(8-n)     ! The taskB is invoked on node(n+1:8)
!$xmp   end task
!$xmp end tasks
      end

      subroutine taskA(n)
      real,save :: yours[*],mine[*]
!$xmp nodes subnode(n)=*         ! An executing node array

      me = this_image()
      if(me.eq. NUM_IMAGES())then
         call xmp_get_primary_image_index(1,me,meabs) ! meabs=n.
```

```
            call exchange(yours,mine,meabs,meabs+1,NUM_IMAGES())
         endif
         sync all                  ! Synchronization within node(1:n).
         if(me.ne.NUM_IMAGES())mine = mine[NUM_IMAGES()]
         sync all                  ! Synchronization within node(1:n).
         end

         subroutine taskB(m)
         real,save :: yours[*],mine[*]
   !$xmp nodes subnode(m)=*        ! An executing node array

         me = this_image()
         if(me.eq.1)then
            call xmp_get_abs_image_index(1,me,meabs) ! meabs=n+1.
            call exchange(yours,mine,meabs,meabs-1,NUM_IMAGES())
         endif
         sync all                  ! Synchronization within node(n+1:8).
         if(me.ne.1)mine = mine[1]
         sync all                  ! Synchronization within node(n+1:8).
         end

         subroutine exchange(yours,mine,meabs,iyouabs,nnodes)
         USE, INTRINSIC :: ISO_FORTRAN_ENV
         real :: yours[*],mine[*]
         real, save :: s[*]                       ! only for exchage.
         TYPE(LOCK_TYPE),save :: lock[*]          ! for lock.
   !$xmp nodes subnode(nnodes)=*     ! An executing node array.
   !$xmp nodes node(8)=**            ! The coarrays s and lock are
   !$xmp coarray on node :: s,lock   ! always visible on node(1:8).

         LOCK(lock[meabs])   ! node(n) puts yours[n] to s[n] and
         s[meabs] = yours    ! node(n+1) puts yours[n+1] to s[n+1].
         UNLOCK(lock[meabs])

         LOCK(lock[iyouabs])  ! node(n) gets s[n+1] into mine[n] and
         mine = s[iyouabs]    ! node(n+1) gets s[n] into mine[n+1].
         UNLOCK(lock[iyouabs])
         end
```

1.8 LOCAL_ALIAS directive

1.8.1 Purpose and form of the LOCAL_ALIAS directive

The LOCAL_ALIAS directive is defined only in XcalableMP Fortran.

The LOCAL_ALIAS directive associates a non-mapped array with an explicitly mapped array. The non-mapped array is an associating local array and the explicitly mapped array is an associated global array. The rank of the associating local array is the same as that of the associated global array. The shape of the associating local array is the same as that of the node-local portion of the associated global array including shadow area. Each element of the associating local array corresponds in array element order to that of the node-local portion of the associated global array. An associating local array always has the dynamic type and type parameter values of the corresponding associated global array.

An associating local array may be a coarray. An associating local array that is a coarray is an on-node coarray of the node array onto which the corresponding associated global array is mapped. Every specification and restriction on coarrays is also applied to an associating local array that is a coarray except that an associating local array is always declared with *deferred-shape-spec-list* of the same rank as the associated global array. In particular, a processor shall ensure that an associating local array that is a coarray has the same bounds on all the images corresponding to the node array onto which the corresponding associated global array is mapped. The mapping attributes that an associated global array may have are processor-dependent.

image-directive	**is**	`LOCAL_ALIAS` *local-rename-list*
local-rename	**is**	*local-array-name* `=>` *global-array-name*

- A *global-array-name* shall be a name of an explicitly mapped array declared in the same scoping unit.
- A *local-array-name* shall be a name of a non-mapped array declared in the same scoping unit.
- A *local-array-name* shall not be a dummy argument.
- An associating local array shall have the declared type and type parameters of the corresponding associated global array.
- An associating local array shall be declared with *deferred-shape-spec-list* of the same rank as the corresponding associated global array.
- A *local-array-name* shall appear in a COARRAY directive in the same scoping unit and the *node-name* in the COARRAY directive shall be the name of the node array onto which the associated global array is mapped.
- If an associated global array is a dummy argument and corresponds to an associating local array that is a coarray, the corresponding effective argument shall be an explicitly mapped array or a subobject of an explicitly mapped array whose name appears in a LOCAL_ALIAS directive and the corresponding associating local array shall be a coarray.
- If a dummy argument is a coarray and the corresponding ultimate argument is a coarray appearing in a LOCAL_ALIAS directive, the dummy

argument shall appear in a COARRAY directive with a node array corresponding to a subset of the set of images that corresponds to the node array onto which the ultimate argument is mapped.

1.8.2 An example of the LOCAL_ALIAS directive

In the following code fragment, the associating local array `la` corresponds to the associated global array `ga(1:5)` on `node(1)` and the associating local array `la` corresponds to the associated global array `ga(6:10)` on `node(2)`.

```
!$xmp nodes node(1:2)=**
      real, save :: ga(10)
      real, save :: la(:)[*]
!$xmp distribute ga(block) onto node
!$xmp local_alias la=>ga
!$xmp coarray on node :: la
```