Section 7

Storage and Sequence Association

HPF allows the mapping of variables across multiple processors in order to improve parallel performance. FORTRAN 77 and Fortran 90 both specify relationships between the storage for data objects associated through `COMMON` and `EQUIVALENCE` statements, and the order of array elements during association at procedure boundaries between actual arguments and dummy arguments. Otherwise, the location of data is not constrained by the language.

`COMMON` and `EQUIVALENCE` statements constrain the alignment of different data items based on the underlying model of storage units and storage sequences:

> *Storage association is the association of two or more data objects that occurs when two or more storage sequences share or are aligned with one or more storage units.*
>
> — Fortran Standard (14.6.3.1)

The model of storage association is a single linearly addressed memory, based on the traditional single address space, single memory unit architecture. This model can cause severe inefficiencies on architectures where storage for variables is mapped.

Sequence association refers to the order of array elements that Fortran requires when an array expression or array element is associated with a dummy array argument:

> *The rank and shape of the actual argument need not agree with the rank and shape of the dummy argument, ...*
>
> — Fortran Standard (12.4.1.4)

As with storage association, sequence association is a natural concept only in systems with a linearly addressed memory.

As an aid to porting FORTRAN 77 codes, HPF allows codes that rely on sequence and storage association to be valid in HPF. Some modification to existing FORTRAN 77 codes may nevertheless be necessary. This chapter explains the relationship between HPF data mapping and sequence and storage association.

7.1 Storage Association

7.1.1 Definitions

1. `COMMON` blocks are either *sequential* or *nonsequential*, as determined by either explicit directive or compiler default. A sequential `COMMON` block has a single common block storage sequence (5.5.2.1).

2. An *aggregate variable group* is a collection of variables whose individual storage sequences are parts of a single storage sequence.

   Variables associated by EQUIVALENCE statements or by a combination of EQUIVALENCE and COMMON statements form an aggregate variable group. The variables of a sequential COMMON block form a single aggregate variable group.

3. The *size* of an aggregate variable group is the number of storage units in the group's storage sequence (14.6.3.1).

4. If there is a member in an aggregate variable group whose storage sequence is totally associated (14.6.3.3) with the storage sequence of the aggregate variable group, that variable is called an *aggregate cover.*

5. Variables are either *sequential* or *nonsequential.* A variable is *sequential* if and only if any of the following holds:

   (a) it appears in a sequential COMMON block;
   (b) it is a member of an aggregate variable group;
   (c) it is an assumed-size array;
   (d) it is a component of a derived type with the Fortran 90 SEQUENCE attribute; or
   (e) it is declared to be sequential in an HPF SEQUENCE directive.

   A sequential variable can be storage associated or sequence associated; nonsequential variables cannot.

6. A COMMON block contains a sequence of *components.* Each component is either an aggregate variable group, or a variable that is not a member of any aggregate variable group. Sequential COMMON blocks contain a single component. Nonsequential COMMON blocks may contain several components that may be nonsequential or sequential variables or aggregate variable groups.

7. A variable is *explicitly mapped* if it appears in an HPF alignment or distribution directive within the scoping unit in which it is declared; otherwise it is *implicitly mapped.*

7.1.2 Examples of Definitions

```
      IMPLICIT REAL (A-Z)
      COMMON /FOO/ A(100), B(100), C(100), D(100), E(100)
      DIMENSION X(100), Y(150), Z(200)


!Example 1:
      EQUIVALENCE ( A(1), Z(1) )
!Four components: (A, B), C, D, E
!Sizes are: 200, 100, 100, 100

!Example 2:
      EQUIVALENCE ( B(100), Y(1) )
```

```
!Three components A, (B, C, D), E
!Sizes are: 100, 300, 100

!Example 3:
      EQUIVALENCE ( E(1), Y(1) )
!Five components: A, B, C, D, E
!Sizes are: 100, 100, 100, 100, 150

!Example 4:
      EQUIVALENCE ( A(51), X(1) ) ( B(100), Y(1) )
!Two components (A, B, C, D), E
!Sizes are: 400, 100

!Example 5:
      EQUIVALENCE ( A(51), X(1) ) ( C(80), Y(1) )
!Two components: (A, B), (C, D, E)
!Sizes are: 200, 300

!Example 6:
      EQUIVALENCE (Y(100), Z(1))
!One aggregate variable group (Y, Z), not involving the COMMON block.
!Size is 299

!Example 7:
!HPF$ SEQUENCE /FOO/
!The COMMON has one component, (A, B, C, D, E)
!Size is 500
```

In Examples 1–6, COMMON block /FOO/ is nonsequential. Aggregate variable groups are shown as components in parentheses. Aggregate covers are Z in Example 1 and Y in Example 3.

7.1.3 Sequence Directives

A SEQUENCE directive is defined to allow a user to declare explicitly that variables or COMMON blocks are to be treated by the compiler as sequential. (COMMON blocks are by default nonsequential. Variables are nonsequential unless Definition 5 applies.) Some implementations may supply an optional compilation environment where the SEQUENCE directive is applied by default. For completeness in such an environment, HPF defines a NO SEQUENCE directive to allow a user to establish that the usual nonsequential default should apply to a scoping unit, or selected variables and COMMON blocks within the scoping unit.

H701	*sequence-directive*	**is**	SEQUENCE [[::] *association-name-list*]
		or	NO SEQUENCE [[::] *association-name-list*]
H702	*association-name*	**is**	*variable-name*
		or	/ *common-block-name* /

Constraint: The result variable of an array-valued function that is not an intrinsic function is a nonsequential array. It may not appear in any HPF SEQUENCE directive.

Constraint: A variable or `COMMON` block name may appear at most once in a *sequence-directive* within any scoping unit.

7.1.4 Storage Association Rules

1. A *sequence-directive* with an empty *association-name-list* is treated as if it contained the name of all implicitly mapped variables and `COMMON` blocks in the scoping unit which cannot otherwise be determined to be sequential or nonsequential by their language context.

2. A sequential variable may not be explicitely mapped unless it is a scalar or rank-one array that is an aggregate cover. If there is more than one aggregate cover for an aggregate variable group, only one may be explicitly mapped.

3. No explicit mapping may be given for a component of a derived type having the Fortran 90 `SEQUENCE` attribute.

4. If a `COMMON` block is nonsequential, then all of the following must hold:

   (a) Every occurrence of the `COMMON` block has exactly the same number of components with each corresponding component having a storage sequence of exactly the same size;

   (b) If a component is a nonsequential variable in *any* occurrence of the `COMMON` block, then it must be nonsequential with identical type, shape, and mapping attributes in *every* occurrence of the `COMMON` block;

   (c) If a component is sequential and explicitly mapped (either a variable or an aggregate variable group with an explicitly mapped aggregate cover) in any occurrence of the `COMMON` block, then it must be sequential and explicitly mapped with identical mapping attributes in *every* occurrence of the `COMMON` block. In addition, the type and shape of the explicitly mapped variable must be identical in all occurrences; and

   (d) Every occurrence of the `COMMON` block must be nonsequential.

7.1.5 Storage Association Discussion

Advice to users. Under these rules, variables in a `COMMON` block can be mapped as long as the components of the `COMMON` block are the same in every scoping unit that declares the `COMMON` block. Rules 4 and 5 also allow variables involved in an `EQUIVALENCE` statement to be mapped by the mechanism of declaring a rank-one array to cover exactly the aggregate variable group and mapping that array.

Since an HPF program is nonconforming if it specifies any mapping that would cause a scalar data object to be mapped onto more than one abstract processor, there is a constraint on the sequential variables and aggregate covers that can be mapped. In particular, programs that direct double precision or complex arrays to be mapped such that the storage units of a single array element are split because of some `EQUIVALENCE` statement or `COMMON` block layout are nonconforming.

Correct FORTRAN 77 or Fortran 90 programs will not necessarily be correct without modification in HPF. As the examples in the next section illustrate, use of

EQUIVALENCE with COMMON blocks can impact mappability of the variables in subtle ways. To allow maximum optimization for performance, the HPF default for variables is to consider them mappable. In order to get correct separate compilation for subprograms that use COMMON blocks with different aggregate variable groups in different scoping units, it will be necessary to insert the HPF SEQUENCE directive.

As a check-list for a user to determine the status of a variable or COMMON block, the following questions can be applied, in order:

- Does the variable appear in some explicit language context which dictates sequential (e.g. EQUIVALENCE) or nonsequential (e.g. array-valued function result variable)?
- If not, does the variable appear in an explicit mapping directive?
- If not, does the variable or COMMON block name appear in the list of names on a SEQENCE or NO SEQUENCE directive?
- If not, does the scoping unit contain a nameless SEQUENCE or NO SEQUENCE?
- If not, is the compilation affected by some special implementation-dependent environment which dictates that names default to SEQUENCE?
- If not, then the compiler will consider the variable or COMMON block name nonsequential and is free to apply data mapping optimizations disregarding Fortran sequence and storage association.

(*End of advice to users.*)

Advice to implementors. In order to protect the user and to facilitate portability of older codes, two implementation options are strongly recommended. First, every implementation should supply some mechanism to verify that the type and shape of every mappable array and the sizes of aggregate variable groups in COMMON blocks are the same in every scoping unit unless the COMMON blocks are declared to be sequential. This same check should also verify that identical mappings have been selected for the variables in COMMON blocks. Implementations without interprocedural information can use a link-time check. The second implementation option recommended is a mechanism to declare that variables and COMMON blocks for a given compilation should be considered sequential unless declared otherwise. The purpose of this feature is to permit compilation of large old libraries or subprograms where storage association is known to exist without requiring that the code be modified to apply the HPF SEQUENCE directive to every COMMON block. (*End of advice to implementors.*)

7.1.6 Examples of Storage Association

```
      IMPLICIT REAL (A-Z)
      COMMON /FOO/ A(100), B(100), C(100), D(100), E(100)
      DIMENSION X(100), Y(150), Z(200), ZZ(300)

      EQUIVALENCE ( A(1), Y(1) )
!Aggregate variable group is not mappable.
!Sizes are: 200, 100, 100, 100.
```

```
      EQUIVALENCE ( B(100), Y(1) ), ( B(1), ZZ(1) )
!Aggregate variable group is mappable only by mapping ZZ.
!ZZ is an aggregate cover for B, C, D, and Y.
!Sizes are: 100, 300, 100.

      EQUIVALENCE ( E(1), Y(1) )
!Aggregate variable group is mappable by mapping Y.
!Sizes are: 100, 100, 100, 100, 150.

      COMMON /TWO/ A(20,40),E(10,10),G(10,100,1000),H(100),P(100)
      REAL COVER(200)
      EQUIVALENCE (COVER(1), H(1))
!HPF$   SEQUENCE A
!HPF$   ALIGN E ...
!HPF$   DISTRIBUTE COVER (CYCLIC(2))
```

Here A is sequential and implicitly mapped, E is explicitly mapped, G is implicitly mapped, the aggregate cover of the aggregate variable group (H, P) is explicitly mapped. /TWO/ is a nonsequential COMMON block.

In another subprogram, the following declarations may occur:

```
      COMMON /TWO/ A(800), E(10,10), G(10,100,1000), Z(200)
!HPF$   SEQUENCE A, Z
!HPF$   ALIGN E ...
!HPF$   DISTRIBUTE Z (CYCLIC(2))
```

There are four components of the same size in both occurrences. Components one and four are sequential. Components two and four are explicitly mapped, with the same type, shape and mapping attributes.

The first component, A, must be declared sequential in both occurrences because its shape is different. It may not be explicitly mapped in either because it is not rank-one or scalar in the first.

E and G must agree in type and shape in both occurrences. E must have the same explicit mapping and G must have no explicit mapping in both occurrences, since they are nonsequential variables.

The fourth component must have the same explicit mapping in both occurrences, and must be made sequential explicitly in the second.

7.2 Argument Passing and Sequence Association

For actual arguments in a procedure call, Fortran 90 allows an array element (scalar) to be associated with a dummy argument that is an array. It furthermore allows the shape of a dummy argument to differ from the shape of the corresponding actual array argument, in effect reshaping the actual argument via the subroutine call. Storage sequence properties of Fortran are used to identify the values of the dummy argument. This feature, carried over from FORTRAN 77, has been widely used to pass starting addresses of subarrays, rows or columns of a larger array, to procedures. For HPF arrays that are potentially mapped across processors, this feature is not fully supported.

7.2.1 Sequence Association Rules

1. When an array element or the name of an assumed-size array is used as an actual argument, the associated dummy argument must be a scalar or specified to be a sequential array.

   An array-element designator of a nonsequential array must not be associated with a dummy array argument.

2. When an actual argument is an array or array section and the corresponding dummy argument differs from the actual argument in shape, then the dummy argument must be declared sequential and the actual array argument must be sequential.

3. A variable of type character (scalar or array) is nonsequential if it conforms to the requirements of Definition 5 of Section 7.1.1. If the length of an explicit-length character dummy argument differs from the length of the actual argument, then both the actual and dummy arguments must be sequential.

7.2.2 Discussion of Sequence Association

When the shape of the dummy array argument and its associated actual array argument differ, the actual argument must not be an expression. There is no HPF mechanism for declaring that the value of an array-valued expression is sequential. In order to associate such an expression as an actual argument with a dummy argument of different rank, the actual argument must first be assigned to a named array variable that is forced to be sequential according to Definition 5 of Section 7.1.1.

7.2.3 Examples of Sequence Association

Given the following subroutine fragment:

```
      SUBROUTINE HOME (X)
      DIMENSION X (20,10)
```

By rule 1

```
      CALL HOME (ET (2,1))
```

is legal only if X is declared sequential in HOME and ET is sequential in the calling routine.

Likewise, by rule 2

```
      CALL HOME (ET)
```

requires either that ET and X are both sequential arrays or that ET is dimensioned exactly the same as X.

Rule 3 addresses a special consideration for variables of type character. Change of the length of character variables across a call, as in

```
      CHARACTER (LEN=44) one_long_word
      one_long_word = 'Chargoggagoggmanchaugagoggchaubunagungamaugg'
      CALL webster(one_long_word)

      SUBROUTINE webster(short_dictionary)
      CHARACTER (LEN=4) short_dictionary (11)
          !Note that short_dictionary(3) is 'agog', for example
```