Section 5

Intrinsic and Library Procedures

HPF includes Fortran 90’s intrinsic procedures. It also adds new intrinsic procedures in two categories: system inquiry intrinsic functions and computational intrinsic functions.

The definitions of two Fortran 90 intrinsic functions, MAXLOC and MINLOC, are extended by the addition of an optional DIM argument.

In addition to the new intrinsic functions, HPF defines a library module, HPF_LIBRARY, that must be provided by vendors of any full HPF implementation.

This description of HPF intrinsic and library procedures follows the form and conventions of Section 13 of the Fortran 90 standard. The material of Sections 13.1, 13.2, 13.3, 13.5.7, 13.8.1, 13.8.2, 13.9, and 13.10 is applicable to the HPF intrinsic and library procedures and to their descriptions in this section of the HPF document.

5.1 Notation

In the examples of this section, T and F are used to denote the logical values true and false.

5.2 System Inquiry Intrinsic Functions

In a multi-processor implementation, the processors may be arranged in an implementation-dependent multi-dimensional processor array. The system inquiry functions return values related to this underlying machine and processor configuration, including the size and shape of the underlying processor array. NUMBER_OF_PROCESSORS returns the total number of processors available to the program or the number of processors available to the program along a specified dimension of the processor array. PROCESSORS_SHAPE returns the shape of the processor array.

The values returned by the system inquiry intrinsic functions remain constant for the duration of one program execution. Thus, NUMBER_OF_PROCESSORS and PROCESSORS_SHAPE have values that are restricted expressions and may be used wherever any other Fortran 90 restricted expression may be used. In particular, NUMBER_OF_PROCESSORS may be used in a specification expression.

The values of system inquiry functions may not occur in initialization expressions, because they may not be assumed to be constants. In particular, HPF programs may be compiled to run on machines whose configurations are not known at compile time.

Note that the system inquiry functions query the physical machine, and have nothing to do with any PROCESSORS directive that may occur.
Advice to users. SIZE(PROCESSORS\_SHAPE()) returns the rank of the processor array. References to system inquiry functions may occur in array declarations and in HPF directives, as in:

\[
\text{INTEGER, DIMENSION(SIZE(PROCESSORS\_SHAPE())) :: PSHAPE}
\]

HPFS $\text{T}(100, 3\times \text{NUMBER\_OF\_PROCESSORS}())$

(End of advice to users.)

5.3 Computational Intrinsic Functions

HPF adds one new intrinsic function, ILEN, which computes the number of bits needed to store an integer value. HPF also generalizes the Fortran 90 MAXLOC and MINLOC intrinsic functions with an optional DIM parameter, for finding the locations of maximum or minimum elements along a given dimension.

5.4 Library Procedures

The mapping inquiry subroutines and computational functions described in this section are available in the HPF library module, HPF\_LIBRARY. Use of these procedures must be accompanied by an appropriate USE statement in each scoping unit in which they are used. They are not intrinsic.

5.4.1 Mapping Inquiry Subroutines

HPF provides data mapping directives that are advisory in nature. The mapping inquiry subroutines allow the program to determine the actual mapping of an array at run time. It may be especially important to know the exact mapping when an EXTRINSIC subprogram is invoked. For these reasons, HPF includes mapping inquiry subroutines which describe how an array is actually mapped onto a machine. To keep the number of routines small, the inquiry procedures are structured as subroutines with optional INTENT (OUT) arguments.

5.4.2 Bit Manipulation Functions

The HPF library includes three elemental bit-manipulation functions. LEADZ computes the number of leading zero bits in an integer’s representation. POPCNT counts the number of one bits in an integer. POPPAR computes the parity of an integer.

5.4.3 Array Reduction Functions

HPF adds additional array reduction functions that operate in the same manner as the Fortran 90 SUM and ANY intrinsic functions. The new reduction functions are IALL, IANY, IPARITY, and PARITY, which correspond to the commutative, associative binary operations IAND, IOR, IERO, and .NEQV., respectively.

In the specifications of these functions, the terms “XXX reduction” are used, where XXX is one of the binary operators above. These are defined by means of an example. The IAND reduction of all the elements of array for which the corresponding element of mask is true is the scalar integer computed in result by
result = IAND_IDENTITY_ELEMENT
DO i_1 = LBOUND(array,1), UBOUND(array,1)
  ... 
  DO i_n = LBOUND(array,n), UBOUND(array,n)
  IF ( mask(i_1,i_2,...,i_n) ) &
    result = IAND( result, array(i_1,i_2,...,i_n) )
  END DO
  ... 
END DO

Here, \( n \) is the rank of \( \text{array} \) and \( \text{IAND_IDENTITY_ELEMENT} \) is the integer which has all bits equal to one. (The interpretation of an integer as a sequence of bits is given in Section 13.5.7 of the Fortran 90 standard.) The other three reductions are similarly defined. The identity elements for \( \text{IOR} \) and \( \text{IEOR} \) are zero. The identity element for \( \text{PARITY} \) is \( .FALSE. \).

### 5.4.4 Array Combining Scatter Functions

These are all generalized array reduction functions in which completely general, but nonoverlapping, subsets of array elements can be combined. There is a corresponding scatter function for each of the twelve reduction operation in the language. The way the elements of the source array are associated with the elements of the result is described in this section; the method of combining their values is described in the specifications of the individual functions in Section 5.7.

These functions all have the form

\[
\text{XXX_SCATTER(ARRAY, BASE, INDX1, ..., INDXn, MASK)}
\]

The allowed values of \( \text{XXX} \) are \( \text{ALL, ANY, COPY, COUNT, IALL, IANY, IPARITY, MAXVAL, MINVAL, PARITY, PRODUCT, and SUM} \). The number of \( \text{INDX} \) arguments must equal the rank of \( \text{BASE} \). Except for \( \text{COUNT_SCATTER} \), \( \text{ARRAY} \) and \( \text{BASE} \) are arrays of the same type. For \( \text{COUNT_SCATTER} \), \( \text{ARRAY} \) is of type logical and \( \text{BASE} \) is of type integer. The argument \( \text{MASK} \) is logical, and the \( \text{INDX} \) arrays are integer. \( \text{ARRAY, MASK} \), and all the \( \text{INDX} \) arrays are conformable. \( \text{MASK} \) is optional. (For \( \text{ALL_SCATTER, ANY_SCATTER, COUNT_SCATTER, and PARITY_SCATTER} \), the \( \text{ARRAY} \) must be logical. These functions do not have an optional \( \text{MASK} \) argument. To conform with the conventions of the F90 standard, the required \( \text{ARRAY} \) argument to these functions is called \( \text{MASK} \) in their specifications in Section 5.7.) The result has the same type, kind type parameter, and shape as \( \text{BASE} \).

For every element \( a \) in \( \text{ARRAY} \) there is a corresponding element in each of the \( \text{INDX} \) arrays. Let \( s_1 \) be the value of the element of \( \text{INDX1} \) that is indexed by the same subscripts as element \( a \) of \( \text{ARRAY} \). More generally, for each \( j = 1, 2, ..., n \), let \( s_j \) be the value of the element of \( \text{INDXj} \) that corresponds to element \( a \) in \( \text{ARRAY} \), where \( n \) is the rank of \( \text{BASE} \). The integers \( s_j, j = 1, ..., n \), form a subscript selecting an element of \( \text{BASE} \): \( \text{BASE}(s_1, s_2, ..., s_n) \).

Thus the \( \text{INDX} \) arrays establish a mapping from all the elements of \( \text{ARRAY} \) onto selected elements of \( \text{BASE} \). Viewed in the other direction, this mapping associates with each element \( b \) of \( \text{BASE} \) a set \( S \) of elements from \( \text{ARRAY} \).

Because \( \text{BASE} \) and the result are conformable, for each element of \( \text{BASE} \) there is a corresponding element of the result.
SECTION 5. INTRINSIC AND LIBRARY PROCEDURES

If \( S \) is empty, then the element of the result corresponding to the element \( b \) of BASE has the same value as \( b \).

If \( S \) is non-empty, then the elements of \( S \) will be combined with element \( b \) to produce an element of the result. The particular means of combining these values is described in the result value section of the specification of the routine below. As an example, for SUM_SCATTER, if the elements of \( S \) are \( a_1, \ldots, a_m \), then the element of the result corresponding to the element \( b \) of BASE is the result of evaluating \( \text{SUM}(//a_1, a_2, \ldots, a_m, b/) \).

Note that, since a scalar is conformable with any array, a scalar may be used in place of an INDX array, in which case one hyperplane of the result is selected. See the example below.

If the optional, final MASK argument is present, then only the elements of ARRAY in positions for which MASK is true participate in the operation. All other elements of ARRAY and of the INDX arrays are ignored and cannot have any influence on any element of the result.

For example, if

\[
\begin{align*}
\text{A is the array} & & \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}; & & \text{B is the array} & & \begin{bmatrix} -1 & -2 & -3 \\ -4 & -5 & -6 \\ -7 & -8 & -9 \end{bmatrix}; \\
\text{I1 is the array} & & \begin{bmatrix} 1 & 1 & 1 \end{bmatrix}; & & \text{I2 is the array} & & \begin{bmatrix} 1 & 2 & 3 \\ 1 & 1 & 2 \\ 1 & 1 & 1 \end{bmatrix}
\end{align*}
\]

then

\[
\begin{align*}
\text{SUM_SCATTER}(\text{A, B, I1, I2}) & = \begin{bmatrix} 14 & 6 & 0 \\ 8 & -5 & -6 \\ 0 & -8 & -9 \end{bmatrix}; \\
\text{SUM_SCATTER}(\text{A, B, 2, I2}) & = \begin{bmatrix} -1 & -2 & -3 \\ 30 & 3 & -3 \\ -7 & -8 & -9 \end{bmatrix}; \\
\text{SUM_SCATTER}(\text{A, B, I1, 2}) & = \begin{bmatrix} -1 & -2 & -3 \\ -4 & 7 & -6 \\ -7 & -1 & -9 \end{bmatrix}; \\
\text{SUM_SCATTER}(\text{A, B, 2, 2}) & = \begin{bmatrix} -1 & -2 & -3 \\ -4 & 40 & -6 \\ -7 & -8 & -9 \end{bmatrix}.
\end{align*}
\]

If \( \text{A} \) is the array \( \begin{bmatrix} 10 & 20 & 30 & 40 & -10 \end{bmatrix} \), \( \text{B} \) is the array \( \begin{bmatrix} 1 & 2 & 3 & 4 \end{bmatrix} \), and \( \text{IND} \) is the array \( \begin{bmatrix} 3 & 2 & 2 & 1 & 1 \end{bmatrix} \),

then \( \text{SUM_SCATTER}(\text{A, B, IND, MASK=(A .GT. 0)}) \) is \( \begin{bmatrix} 41 & 52 & 13 & 4 \end{bmatrix} \).

5.4.5 Array Prefix and Suffix Functions

In a scan of a vector, each element of the result is a function of the elements of the vector that precede it (for a prefix scan) or that follow it (for a suffix scan). These functions provide scan operations on arrays and subarrays. The functions all have the form
5.4. LIBRARY PROCEDURES

XXX_PREFIX(ARRAY, DIM, MASK, SEGMENT, EXCLUSIVE)
XXX_SUFFIX(ARRAY, DIM, MASK, SEGMENT, EXCLUSIVE)

The allowed values of XXX are ALL, ANY, COPY, COUNT, IALL, IANY, IPARI TY, MAXVAL, MINVAL, PARITY, PRODUCT, and SUM.

When comments below apply to both prefix and suffix forms of the routines, we will refer to them as YYYYFIX functions.

The arguments DIM, MASK, SEGMENT, and EXCLUSIVE are optional. The COPY_YYYYFIX functions do not have MASK or EXCLUSIVE arguments. The ALL_YYYYFIX, ANY_YYYYFIX, COUNT_-
YYYYFIX, and PARITY_YYYYFIX functions do not have MASK arguments. Their ARRAY argument must be of type logical; it is denoted MASK in their specifications in Section 5.7.

The arguments MASK and SEGMENT must be of type logical. SEGMENT must have the same shape as ARRAY. MASK must be conformable with ARRAY. EXCLUSIVE is a logical scalar. DIM is a scalar integer between one and the rank of ARRAY.

Result Value. The result has the same shape as ARRAY, and, with the exception of COUNT_YYYYFIX, the same type and kind type parameter as ARRAY. (The result of COUNT_YYYYFIX is default integer.)

In every case, every element of the result is determined by the values of certain selected elements of ARRAY in a way that is specific to the particular function and is described in its specification. The optional arguments affect the selection of elements of ARRAY for each element of the result; the selected elements of ARRAY are said to contribute to the result element. This section describes fully which elements of ARRAY contribute to a given element of the result.

If no elements of ARRAY are selected for a given element of the result, that result element is set to a default value that is specific to the particular function and is described in its specification.

For any given element r of the result, let a be the corresponding element of ARRAY. Every element of ARRAY contributes to r unless disqualified by one of the following rules.

1. If the function is XXX_PREFIX, no element that follows a in the array element ordering of ARRAY contributes to r. If the function is XXX_SUFFIX, no element that precedes a in the array element ordering of ARRAY contributes to r.

2. If the DIM argument is provided, an element z of ARRAY does not contribute to r unless all its indices, excepting only the index for dimension DIM, are the same as the corresponding indices of a. (It follows that if the DIM argument is omitted, then ARRAY, MASK, and SEGMENT are processed in array element order, as if temporarily regarded as rank-one arrays. If the DIM argument is present, then a family of completely independent scan operations are carried out along the selected dimension of ARRAY.)

3. If the MASK argument is provided, an element z of ARRAY contributes to r only if the element of MASK corresponding to z is true. (It follows that array elements corresponding to positions where the MASK is false do not contribute anywhere to the result. However, the result is nevertheless defined at all positions, even positions where the MASK is false.)
4. If the SEGMENT argument is provided, an element $z$ of ARRAY does not contribute if there is some intermediate element $w$ of ARRAY, possibly $z$ itself, with all of the following properties:

(a) If the function is XXX.PREFIX, $w$ does not precede $z$ but does precede $a$ in the array element ordering; if the function is XXX.SUFFIX, $w$ does not follow $z$ but does follow $a$ in the array element ordering;

(b) If the DIM argument is present, all the indices of $w$, excepting only the index for dimension DIM, are the same as the corresponding indices of $a$; and

(c) The element of SEGMENT corresponding to $w$ does not have the same value as the element of SEGMENT corresponding to $a$. (In other words, $z$ can contribute only if there is an unbroken string of SEGMENT values, all alike, extending from $z$ through $a$.)

5. If the EXCLUSIVE argument is provided and is true, then $a$ itself does not contribute to $r$.

These general rules lead to the following important cases:

Case (i): If ARRAY has rank one, element $i$ of the result of XXX.PREFIX(ARRAY) is determined by the first $i$ elements of ARRAY; element SIZE(ARRAY) $- i = 1$ of the result of XXX.SUFFIX(ARRAY) is determined by the last $i$ elements of ARRAY.

Case (ii): If ARRAY has rank greater than one, then each element of the result of XXX.PREFIX(ARRAY) has a value determined by the corresponding element $a$ of the ARRAY and all elements of ARRAY that precede $a$ in array element order. For XXX.SUFFIX, $a$ is determined by the elements of ARRAY that correspond to or follow $a$ in array element order.

Case (iii): Each element of the result of XXX.PREFIX(ARRAY,MASK=MASK) is determined by selected elements of ARRAY, namely the corresponding element $a$ of the ARRAY and all elements of ARRAY that precede $a$ in array element order, but an element of ARRAY may contribute to the result only if the corresponding element of MASK is true. If this restriction results in selecting no array elements to contribute to some element of the result, then that element of the result is set to the default value for the given function.

Case (iv): Each element of the result of XXX.PREFIX(ARRAY,DIM=DIM) is determined by selected elements of ARRAY, namely the corresponding element $a$ of the ARRAY and all elements of ARRAY that precede $a$ along dimension DIM; for example, in SUM.PREFIX(A(1:N,1:N), DIM=2), result element $(i_1,i_2)$ could be computed as SUM(A(i_1,1 : i_2)). More generally, in SUM.PREFIX(ARRAY, DIM), result element $i_1,i_2,...,i_{DIM},...,i_n$ could be computed as SUM(ARRAY( i_1,i_2,...,i_{DIM},...,i_n )). (Note the colon before i_{DIM} in that last expression.)

Case (v): If ARRAY has rank one, then element $i$ of the result of XXX.PREFIX(ARRAY, EXCLUSIVE=.TRUE.) is determined by the first $i - 1$ elements of ARRAY.

Case (vi): The options may be used in any combination.
Advice to users. A new segment begins at every transition from false to true or true to false; thus a segment is indicated by a maximal contiguous subsequence of like logical values:


----- - - ----- - --- - seven segments

(End of advice to users.)

Rationale.

One existing library delimits the segments by indicating the start of each segment. Another delimits the segments by indicating the stop of each segment. Each method has its advantages. There is also the question of whether this convention should change when performing a suffix rather than a prefix. HPF adopts the symmetric representation above. The main advantages of this representation are:

(A) It is symmetrical, in that the same segment specifier may be meaningfully used for prefix and suffix without changing its interpretation (start versus stop).

(B) The start-bit or stop-bit representation is easily converted to this form by using PARITY_PREFIX or PARITY_SUFFIX. These might be standard idioms for a compiler to recognize:

- \(\text{SUM_PREFIX}(\text{FOO}, \text{SEGMENT} = \text{PARITY_PREFIX}(\text{START_BITS}))\)
- \(\text{SUM_PREFIX}(\text{FOO}, \text{SEGMENT} = \text{PARITY_SUFFIX}(\text{START_BITS}))\)
- \(\text{SUM_SUFFIX}(\text{FOO}, \text{SEGMENT} = \text{PARITY_SUFFIX}(\text{START_BITS}))\)
- \(\text{SUM_SUFFIX}(\text{FOO}, \text{SEGMENT} = \text{PARITY_PREFIX}(\text{STOP_BITS}))\)

(End of rationale.)

Examples. The examples below illustrate all possible combinations of optional arguments for SUM_PREFIX. The default value for SUM_YYYYFIX is zero.

Case (i): \(\text{SUM_PREFIX}(/1,3,5,7/)\) is \[1 \ 4 \ 9 \ 16\].

Case (ii): If \(B\) is the array \[
\begin{bmatrix}
1 & 2 & 3 \\
4 & 5 & 6 \\
7 & 8 & 9
\end{bmatrix}
\]

\(\text{SUM_PREFIX}(B)\) is the array \[
\begin{bmatrix}
1 & 14 & 30 \\
5 & 19 & 36 \\
12 & 27 & 45
\end{bmatrix}
\]

Case (iii): If \(A\) is the array \[
\begin{bmatrix}
3 & 5 & -2 & -1 & 7 & 4 & 8
\end{bmatrix}
\]

then \(\text{SUM_PREFIX}(A, \text{MASK} = A \cdot \text{LT.} \ 6)\) is \[
\begin{bmatrix}
3 & 8 & 6 & 5 & 5 & 9 & 9
\end{bmatrix}
\].

Case (iv): If \(B\) is the array \[
\begin{bmatrix}
1 & 2 & 3 \\
4 & 5 & 6 \\
7 & 8 & 9
\end{bmatrix}
\]

then \(\text{SUM_PREFIX}(B, \text{DIM}=1)\) is the array \[
\begin{bmatrix}
1 & 3 & 6 \\
5 & 7 & 9 \\
12 & 15 & 18
\end{bmatrix}
\]

and \(\text{SUM_PREFIX}(B, \text{DIM}=2)\) is the array \[
\begin{bmatrix}
1 & 3 & 6 \\
4 & 9 & 15 \\
7 & 15 & 24
\end{bmatrix}
\].
Case (v): \texttt{SUM\_PREFIX}((/1,3,5,7/), EXCLUSIVE=.TRUE.) is \[ \begin{bmatrix} 0 & 1 & 4 & 9 \end{bmatrix} \].

Case (vi): If \( B \) is the array \[
\begin{bmatrix}
1 & 2 & 3 & 4 & 5 \\
6 & 7 & 8 & 9 & 10 \\
11 & 12 & 13 & 14 & 15
\end{bmatrix},
\]
and \( S \) is the array \[
\begin{bmatrix}
T & T & F & F & F \\
F & F & T & T & T \\
T & F & F & F & F
\end{bmatrix},
\]
then:

\texttt{SUM\_PREFIX}(B, DIM=2, MASK=M, SEGMENT=S, EXCLUSIVE=.TRUE.) is
\[
\begin{bmatrix}
0 & 1 & 0 & 3 & 7 \\
0 & 0 & 0 & 0 & 9 \\
0 & 11 & 11 & 24 & 24
\end{bmatrix}
\]

\texttt{SUM\_PREFIX}(B, DIM=2, MASK=M, SEGMENT=S, EXCLUSIVE=.FALSE.) is
\[
\begin{bmatrix}
1 & 3 & 3 & 7 & 12 \\
0 & 0 & 8 & 9 & 19 \\
11 & 11 & 24 & 24 & 24
\end{bmatrix}
\]

\texttt{SUM\_PREFIX}(B, DIM=2, MASK=M, EXCLUSIVE=.TRUE.) is
\[
\begin{bmatrix}
0 & 0 & 0 & 8 & 17 \\
0 & 11 & 11 & 24 & 24
\end{bmatrix}
\]

\texttt{SUM\_PREFIX}(B, DIM=2, MASK=M, EXCLUSIVE=.FALSE.) is
\[
\begin{bmatrix}
1 & 3 & 6 & 10 & 15 \\
0 & 0 & 8 & 17 & 27 \\
11 & 11 & 24 & 24 & 24
\end{bmatrix}
\]

\texttt{SUM\_PREFIX}(B, DIM=2, SEGMENT=S, EXCLUSIVE=.TRUE.) is
\[
\begin{bmatrix}
0 & 1 & 0 & 3 & 7 \\
0 & 0 & 7 & 0 & 9 \\
0 & 11 & 23 & 36 & 50
\end{bmatrix}
\]

\texttt{SUM\_PREFIX}(B, DIM=2, SEGMENT=S, EXCLUSIVE=.FALSE.) is
\[
\begin{bmatrix}
1 & 3 & 3 & 7 & 12 \\
6 & 7 & 15 & 9 & 19 \\
11 & 23 & 36 & 50 & 65
\end{bmatrix}
\]

\texttt{SUM\_PREFIX}(B, DIM=2, EXCLUSIVE=.TRUE.) is
\[
\begin{bmatrix}
0 & 6 & 13 & 21 & 30 \\
0 & 11 & 23 & 36 & 50
\end{bmatrix}
\]

\texttt{SUM\_PREFIX}(B, DIM=2, EXCLUSIVE=.FALSE.) is
\[
\begin{bmatrix}
1 & 3 & 6 & 10 & 15 \\
6 & 13 & 21 & 30 & 40 \\
11 & 23 & 36 & 50 & 65
\end{bmatrix}
\]

\texttt{SUM\_PREFIX}(B, MASK=M, SEGMENT=S, EXCLUSIVE=.TRUE.) is
\[
\begin{bmatrix}
0 & 11 & 0 & 0 & 0 \\
0 & 11 & 0 & 4 & 5 \\
0 & 11 & 8 & 0 & 0
\end{bmatrix}
\]

\texttt{SUM\_PREFIX}(B, MASK=M, SEGMENT=S, EXCLUSIVE=.FALSE.) is
\[
\begin{bmatrix}
1 & 13 & 3 & 4 & 5 \\
0 & 13 & 8 & 13 & 15 \\
11 & 13 & 21 & 0 & 0
\end{bmatrix}
\]

\texttt{SUM\_PREFIX}(B, MASK=M, EXCLUSIVE=.TRUE.) is
\[
\begin{bmatrix}
0 & 12 & 14 & 38 & 51 \\
1 & 14 & 17 & 42 & 56 \\
1 & 14 & 25 & 51 & 66
\end{bmatrix}
\]
SUM_PREFIX(B, MASK=M, EXCLUSIVE=.FALSE.) is
[ 1 14 17 42 56 ]
[ 1 14 25 51 66 ]
[ 12 14 38 51 66 ]

SUM_PREFIX(B, SEGMENT=S, EXCLUSIVE=.TRUE.) is
[ 0 11 0 0 0 ]
[ 0 13 0 4 5 ]
[ 0 20 8 0 0 ]

SUM_PREFIX(B, SEGMENT=S, EXCLUSIVE=.FALSE.) is
[ 1 13 3 4 5 ]
[ 6 20 8 13 15 ]
[ 11 32 21 14 15 ]

SUM_PREFIX(B, EXCLUSIVE=.TRUE.) is
[ 0 18 39 63 90 ]
[ 1 20 42 67 95 ]
[ 7 27 50 76 105 ]

SUM_PREFIX(B, EXCLUSIVE=.FALSE.) is
[ 1 20 42 67 95 ]
[ 7 27 50 76 105 ]
[ 18 39 63 90 120 ]

5.4.6 Array Sorting Functions

HPF includes procedures for sorting multidimensional arrays. These are structured as functions that return sorting permutations. An array can be sorted along a given axis, or the whole array may be viewed as a sequence in array element order. The sorts are stable, allowing for convenient sorting of structures by major and minor keys.

5.5 Generic Intrinsic and Library Procedures

For all of the intrinsic and library procedures, the arguments shown are the names that must be used for keywords when using the keyword form for actual arguments. Many of the argument keywords have names that are indicative of their usage, as is the case in Fortran 90. See Section 13.10 of the standard.

5.5.1 System inquiry intrinsic functions

NUMBER_OF_PROCESSORS(DIM) The number of executing processors
Optional DIM
PROCESSORS_SHAPE() The shape of the executing processor array

5.5.2 Array location intrinsic functions

MAXLOC(ARRAY, DIM, MASK) Location of a maximum value in an array
Optional DIM, MASK
MINLOC(ARRAY, DIM, MASK) Location of a minimum value in an array
Optional DIM, MASK
5.5.3 Mapping inquiry subroutines

HPF_ALIGNMENT(ALIGNEE, LB, UB, STRIDE, AXIS_MAP, IDENTITY_MAP, &
DYNAMIC, NCOPIES)
Optional LB, UB, STRIDE, AXIS_MAP, IDENTITY_MAP, DYNAMIC, NCOPIES
HPF_TEMPLATE(ALIGNEE, TEMPLATE_RANK, LB, UB, AXIS_TYPE, AXIS_INFO, &
NUMBER_ALIGNED, DYNAMIC)
Optional TEMPLATE_RANK, LB, UB, AXIS_TYPE, AXIS_INFO,
NUMBER_ALIGNED, DYNAMIC
HPF_DISTRIBUTION(DISTRIBUTEE, AXIS_TYPE, AXIS_INFO, PROCESSORS_RANK, &
PROCESSORS_SHAPE)
Optional AXIS_TYPE, AXIS_INFO, PROCESSORS_RANK, PROCESSORS_SHAPE

5.5.4 Bit manipulation functions

ILEN(I)                Bit length (intrinsic)
LEADZ(I)               Leading zeros
POPCNT(I)              Number of one bits
POPPAR(I)              Parity

5.5.5 Array reduction functions

IALL(IARRAY, DIM, MASK) Bitwise logical AND reduction
Optional DIM, MASK
IANY(IARRAY, DIM, MASK) Bitwise logical OR reduction
Optional DIM, MASK
IPARITY(IARRAY, DIM, MASK) Bitwise logical EOR reduction
Optional DIM, MASK
PARITY(MASK, DIM) Logical EOR reduction
Optional DIM
5.5.6 Array combining scatter functions

\begin{verbatim}
ALL_SCATTER(MASK, BASE, INDEX1 ..., INDEXn)
ANY_SCATTER(MASK, BASE, INDEX1, ..., INDEXn)
COPY_SCATTER(ARRAY, BASE, INDEX1, ..., INDEXn, MASK)
  Optional MASK
COUNT_SCATTER(ARRAY, BASE, INDEX1, ..., INDEXn, MASK)
  Optional MASK
IALL_SCATTER(ARRAY, BASE, INDEX1, ..., INDEXn, MASK)
  Optional MASK
IANY_SCATTER(ARRAY, BASE, INDEX1, ..., INDEXn, MASK)
  Optional MASK
IPARITY_SCATTER(ARRAY, BASE, INDEX1, ..., INDEXn, MASK)
  Optional MASK
IALL_SCATTER(ARRAY, BASE, INDEX1, ..., INDEXn, MASK)
  Optional MASK
MAXVAL_SCATTER(ARRAY, BASE, INDEX1, ..., INDEXn, MASK)
  Optional MASK
MINVAL_SCATTER(ARRAY, BASE, INDEX1, ..., INDEXn, MASK)
  Optional MASK
PARITY_SORT(MASK, BASE, INDEX1, ..., INDEXn)
PRODUCT_SCATTER(ARRAY, BASE, INDEX1, ..., INDEXn, MASK)
  Optional MASK
SUM_SCATTER(ARRAY, BASE, INDEX1, ..., INDEXn, MASK)
  Optional MASK
\end{verbatim}

5.5.7 Array prefix and suffix functions

\begin{verbatim}
ALL_PREFIX(MASK, DIM, SEGMENT, EXCLUSIVE)
  Optional DIM, SEGMENT, EXCLUSIVE
ALL_SUFFIX(MASK, DIM, SEGMENT, EXCLUSIVE)
  Optional DIM, SEGMENT, EXCLUSIVE
ANY_PREFIX(MASK, DIM, SEGMENT, EXCLUSIVE)
  Optional DIM, SEGMENT, EXCLUSIVE
ANY_SUFFIX(MASK, DIM, SEGMENT, EXCLUSIVE)
  Optional DIM, SEGMENT, EXCLUSIVE
\end{verbatim}
5.5.8 Array sort functions

GRADE_DOWN(ARRAY, DIM)  Permutation that sorts into descending order
    Optional DIM
GRADE_UP(ARRAY, DIM)    Permutation that sorts into ascending order
    Optional DIM
5.6 Specifications of Intrinsic Procedures

5.6.1 ILEN(I)

Description. Returns one less than the length, in bits, of the two’s-complement representation of an integer.

Class. Elemental function.

Argument. I must be of type integer.

Result Type and Type Parameter. Same as I.

Result Value. If I is nonnegative, ILEN(I) has the value \( \lfloor \log_2(I + 1) \rfloor \); if I is negative, ILEN(I) has the value \( \lfloor \log_2(-I) \rfloor \).

Examples. ILEN(4) = 3. ILEN(-4) = 2. \(2^{\ast\ast}\)ILEN(N-1) rounds N up to a power of 2 (for N > 0), whereas \(2^{\ast\ast}(\text{ILEN}(N)-1)\) rounds N down to a power of 2. Compare with LEADZ.

The value returned is one less than the length of the two’s-complement representation of I, as the following explains. The shortest two’s-complement representation of 4 is 0100. The leading zero is the required sign bit. In 3-bit two’s complement, 100 represents -4.

5.6.2 MAXLOC(ARRAY, DIM, MASK)

Optional Arguments. DIM, MASK

Description. Determine the locations of the first elements of ARRAY along dimension DIM having the maximum value of the elements identified by MASK.

Class. Transformational function.

Arguments.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARRAY</td>
<td>must be of type integer or real. It must not be scalar.</td>
</tr>
<tr>
<td>DIM (optional)</td>
<td>must be scalar and of type integer with a value in the range (1 \leq \text{DIM} \leq n), where (n) is the rank of ARRAY. The corresponding actual argument must not be an optional dummy argument.</td>
</tr>
<tr>
<td>MASK (optional)</td>
<td>must be of type logical and must be conformable with ARRAY.</td>
</tr>
</tbody>
</table>

Result Type, Type Parameter, and Shape. The result is of type default integer. If DIM is absent the result is an array of rank one and size equal to the rank of ARRAY; otherwise, the result is an array of rank \(n - 1\) and shape \((d_1, \ldots, d_{\text{DIM}-1}, d_{\text{DIM}+1}, \ldots, d_n)\), where \((d_1, \ldots, d_n)\) is the shape of ARRAY.

Result Value. 
Case (i): The result of executing $S = \text{MAXLOC}(\text{ARRAY}) + \text{LBOUND}(\text{ARRAY}) - 1$ is a rank-one array $S$ of size equal to the rank $n$ of ARRAY. It is such that $\text{ARRAY}(S(1), \ldots, S(n))$ has the maximum value of all the elements of ARRAY. If more than one element has the maximum value, the element whose subscripts are returned is the first such element, taken in array element order. If ARRAY has size zero, the result is processor dependent.

Case (ii): The result of executing $S = \text{MAXLOC}(\text{ARRAY}, \text{MASK}) + \text{LBOUND}(\text{ARRAY}) - 1$ is a rank-one array $S$ of size equal to the rank $n$ of ARRAY. It is such that $\text{ARRAY}(S(1), \ldots, S(n))$ corresponds to a true element of MASK, and has the maximum value of all such elements of ARRAY. If more than one element has the maximum value, the element whose subscripts are returned is the first such element, taken in array element order. If there are no such elements (that is, if ARRAY has size zero or every element of MASK has the value false), the result is processor dependent.

Case (iii): If ARRAY has rank one, the result of $\text{MAXLOC}(\text{ARRAY}, \text{DIM}[\text{,MASK}])$ is a scalar $S$ such that $\text{ARRAY}(S + \text{LBOUND}(\text{ARRAY},1) - 1)$ corresponds to a true element of MASK (if MASK is present) and has the maximum value of all such elements (all elements if MASK is absent). It is the smallest such subscript. Otherwise, the value of element $(s_1, \ldots, s_{\text{DIM}-1}, s_{\text{DIM}+1}, \ldots, s_n)$ of $\text{MAXLOC}(\text{ARRAY}, \text{DIM}[\text{,MASK}])$ is equal to $\text{MAXLOC}(\text{ARRAY}(s_1, \ldots, s_{\text{DIM}-1}, s_{\text{DIM}+1}, \ldots, s_n) [\text{,MASK = MASK}(s_1, \ldots, s_{\text{DIM}-1}, s_{\text{DIM}+1}, \ldots, s_n)])$.

Examples.

Case (i): The value of $\text{MAXLOC}(\langle/5,-9,3/\rangle)$ is $[1]$.

Case (ii): $\text{MAXLOC}(C, \text{MASK = C .LT. 0})$ finds the location of the first element of C that is the maximum of the negative elements.

Case (iii): The value of $\text{MAXLOC}(\langle/5,-9,3/\rangle, \text{DIM}=1)$ is 1. If B is the array

\[
\begin{bmatrix}
1 & 3 & -9 \\
2 & 2 & 6
\end{bmatrix},
\]

$\text{MAXLOC}(B, \text{DIM} = 1)$ is $[2 1 2]$ and $\text{MAXLOC}(B, \text{DIM} = 2)$ is $[2 3]$.

Note that this is true even if B has a declared lower bound other than 1.

5.6.3 MINLOC(ARRAY, DIM, MASK)

Optional Arguments. DIM, MASK

Description. Determine the locations of the first elements of ARRAY along dimension DIM having the minimum value of the elements identified by MASK.

Class. Transformational function.

Arguments.

ARRAY must be of type integer or real. It must not be scalar.
5.6. SPECIFICATIONS OF INTRINSIC PROCEDURES

DIM (optional) must be scalar and of type integer with a value in the range \( 1 \leq \text{DIM} \leq n \), where \( n \) is the rank of ARRAY. The corresponding actual argument must not be an optional dummy argument.

MASK (optional) must be of type logical and must be conformable with ARRAY.

Result Type, Type Parameter, and Shape. The result is of type default integer. If DIM is absent the result is an array of rank one and size equal to the rank of ARRAY; otherwise, the result is an array of rank \( n - 1 \) and shape \((d_1, \ldots, d_{\text{DIM}-1}, d_{\text{DIM}+1}, \ldots, d_n)\), where \((d_1, \ldots, d_n)\) is the shape of ARRAY.

Result Value.

Case (i): The result of executing \( S = \text{MINLOC}(\text{ARRAY}) + \text{LBOUND}(\text{ARRAY}) - 1 \) is a rank-one array \( S \) of size equal to the rank \( n \) of ARRAY. It is such that \( \text{ARRAY}(S(1), \ldots, S(n)) \) has the minimum value of all of the elements of ARRAY. If more than one element has the minimum value, the element whose subscripts are returned is the first such element, taken in array element order. If ARRAY has size zero, the result is processor dependent.

Case (ii): The result of executing \( S = \text{MINLOC}(\text{ARRAY}, \text{MASK}) + \text{LBOUND}(\text{ARRAY}) - 1 \) is a rank-one array \( S \) of size equal to the rank \( n \) of ARRAY. It is such that \( \text{ARRAY}(S(1), \ldots, S(n)) \) corresponds to a true element of MASK, and has the minimum value of all such elements of ARRAY. If more than one element has the minimum value, the element whose subscripts are returned is the first such element, taken in array element order. If there are no such elements (that is, if ARRAY has size zero or every element of MASK has the value false), the result is processor dependent.

Case (iii): If ARRAY has rank one, the result of \( \text{MINLOC}(\text{ARRAY}, \text{DIM}[,\text{MASK}]) \) is a scalar \( S \) such that \( \text{ARRAY}(S + \text{LBOUND}(\text{ARRAY},1) - 1) \) corresponds to a true element of MASK (if MASK is present) and has the minimum value of all such elements (all elements if MASK is absent). It is the smallest such subscript. Otherwise, the value of element \((s_1, \ldots, s_{\text{DIM}-1}, s_{\text{DIM}+1}, \ldots, s_n)\) of \( \text{MINLOC}(\text{ARRAY}, \text{DIM}[,\text{MASK}]) \) is equal to \( \text{MINLOC}(\text{ARRAY}((s_1, \ldots, s_{\text{DIM}-1}, s_{\text{DIM}+1}, \ldots, s_n))[,\text{MASK} = \text{MASK}((s_1, \ldots, s_{\text{DIM}-1}, s_{\text{DIM}+1}, \ldots, s_n))]) \).

Examples.

Case (i): The value of \( \text{MINLOC}((/5, -9, 3 /)) \) is \([2]\).

Case (ii): \( \text{MINLOC}(C, \text{MASK} = C \cdot \text{GT.} 0) \) finds the location of the first element of \( C \) that is the minimum of the positive elements.

Case (iii): The value of \( \text{MINLOC}((/5, -9, 3 /), \text{DIM}=1) \) is 2. If \( B \) is the array

\[
\begin{bmatrix}
1 & 3 & -9 \\
2 & 2 & 6
\end{bmatrix}
\]

and \( \text{MINLOC}(B, \text{DIM} = 1) \) is \([1 2 1]\).

Note that this is true even if \( B \) has a declared lower bound other than 1.
5.6.4 NUMBE...PROCESSORS(DIM)

Optional Argument. DIM

Description. Returns the total number of processors available to the program or the number of processors available to the program along a specified dimension of the processor array.

Class. System inquiry function.

Arguments.

DIM (optional) must be scalar and of type integer with a value in the range 1 ≤ DIM ≤ n where n is the rank of the processor array.

Result Type, Type Parameter, and Shape. Default integer scalar.

Result Value. The result has a value equal to the extent of dimension DIM of the processor-dependent hardware processor array or, if DIM is absent, the total number of elements of the processor-dependent hardware processor array. The result is always greater than zero.

Examples. For a computer with 8192 processors arranged in a 128 by 64 rectangular grid, the value of NUMBER_OF_PROCESSORS() is 8192; the value of NUMBER_OF_PROCESSORS(DIM=1) is 128; and the value of NUMBER_OF_PROCESSORS(DIM=2) is 64. For a single-processor workstation, the value of NUMBER_OF_PROCESSORS() is 1; since the rank of a scalar processor array is zero, no DIM argument may be used.

5.6.5 PROCESSORS_SHAPE()

Description. Returns the shape of the implementation-dependent processor array.

Class. System inquiry function.

Arguments. None

Result Type, Type Parameter, and Shape. The result is a default integer array of rank one whose size is equal to the rank of the implementation-dependent processor array.

Result Value. The value of the result is the shape of the implementation-dependent processor array.

Example. In a computer with 2048 processors arranged in a hypercube, the value of PROCESSORS_SHAPE() is [2,2,2,2,2,2,2,2]. In a computer with 8192 processors arranged in a 128 by 64 rectangular grid, the value of PROCESSORS_SHAPE() is [128,64]. For a single processor workstation, the value of PROCESSORS_SHAPE() is [] (the size-zero array of rank one).
5.7 Specifications of Library Procedures

5.7.1 ALL_PREFIX(MASK, DIM, SEGMENT, EXCLUSIVE)

Optional Arguments. DIM, SEGMENT, EXCLUSIVE

Description. Computes a segmented logical AND scan along dimension DIM of MASK.

Class. Transformational function.

Arguments.

MASK must be of type logical. It must not be scalar.

DIM (optional) must be scalar and of type integer with a value in the range 1 ≤ DIM ≤ n, where n is the rank of MASK.

SEGMENT (optional) must be of type logical and must have the same shape as MASK.

EXCLUSIVE (optional) must be of type logical and must be scalar.

Result Type, Type Parameter, and Shape. Same as MASK.

Result Value. Element r of the result has the value ALL((/ a1,...,am /)) where (a1,...,am) is the (possibly empty) set of elements of MASK selected to contribute to r by the rules stated in Section 5.4.5.

Example. ALL_PREFIX( (/F,F,T,T,T/), SEGMENT= (/F,F,F,T,T/) ) is

[ T F F T T ].

5.7.2 ALL_SCATTER(MASK,BASE,INDX1, ..., INDXn)

Description. Scatters elements of MASK to positions of the result indicated by index arrays INDX1, ..., INDXn. An element of the result is true if and only if the corresponding element of BASE and all elements of MASK scattered to that position are true.

Class. Transformational function.

Arguments.

MASK must be of type logical. It must not be scalar.

BASE must be of type logical with the same kind type parameter as MASK. It must not be scalar.

INDX1,...,INDXn must be of type integer and conformable with MASK. The number of INDX arguments must be equal to the rank of BASE.

Result Type, Type Parameter, and Shape. Same as BASE.
Result Value. The element of the result corresponding to the element $b$ of BASE has the value $\text{ALL}(\langle a_1, a_2, \ldots, a_m, b \rangle)$, where $(a_1, \ldots, a_m)$ are the elements of MASK associated with $b$ as described in Section 5.4.4.

Example. $\text{ALL\_SCATTER}(\langle T, T, T, F, T \rangle, \langle T, T, T \rangle, \langle 1, 1, 2, 2 \rangle)$ is $\begin{bmatrix} T & F & T \end{bmatrix}$.

5.7.3 $\text{ALL\_SUFFIX}($MASK, DIM, SEGMENT, EXCLUSIVE$)$

Optional Arguments. DIM, SEGMENT, EXCLUSIVE

Description. Computes a reverse, segmented logical AND scan along dimension DIM of MASK.

Class. Transformational function.

Arguments.

- MASK must be of type logical. It must not be scalar.
- DIM (optional) must be scalar and of type integer with a value in the range $1 \leq \text{DIM} \leq n$, where $n$ is the rank of MASK.
- SEGMENT (optional) must be of type logical and must have the same shape as MASK.
- EXCLUSIVE (optional) must be of type logical and must be scalar.

Result Type, Type Parameter, and Shape. Same as MASK.

Result Value. Element $r$ of the result has the value $\text{ALL}(\langle a_1, \ldots, a_m \rangle)$ where $(a_1, \ldots, a_m)$ is the (possibly empty) set of elements of MASK selected to contribute to $r$ by the rules stated in Section 5.4.5.

Example. $\text{ALL\_SUFFIX}(\langle T, F, T, T, T \rangle, \text{SEGMENT=} \langle F, F, F, T, T \rangle)$ is $\begin{bmatrix} F & F & T & T & T \end{bmatrix}$.

5.7.4 $\text{ANY\_PREFIX}($MASK, DIM, SEGMENT, EXCLUSIVE$)$

Optional Arguments. DIM, SEGMENT, EXCLUSIVE

Description. Computes a segmented logical OR scan along dimension DIM of MASK.

Class. Transformational function.

Arguments.

- MASK must be of type logical. It must not be scalar.
- DIM (optional) must be scalar and of type integer with a value in the range $1 \leq \text{DIM} \leq n$, where $n$ is the rank of MASK.
- SEGMENT (optional) must be of type logical and must have the same shape as MASK.
EXCLUSIVE (optional) must be of type logical and must be scalar.

**Result Type, Type Parameter, and Shape.** Same as MASK.

**Result Value.** Element \( r \) of the result has the value \( \text{ANY}(\langle a_1, \ldots, a_m \rangle) \) where \( (a_1, \ldots, a_m) \) is the (possibly empty) set of elements of MASK selected to contribute to \( r \) by the rules stated in Section 5.4.5.

**Example.** \( \text{ANY\_PREFIX}(\langle /F,T,F,F,F/F\rangle, \ \text{SEGMENT=}(\langle/F,F,F,T,T/\rangle) \) is
\[
\begin{bmatrix}
F & T & T & F & F
\end{bmatrix}.
\]

### 5.7.5 ANY\_SCATTER(MASK,BASE,INDX1, ..., INDXn)

**Description.** Scatters elements of MASK to positions of the result indicated by index arrays INDX1, ..., INDXn. An element of the result is true if and only if the corresponding element of BASE or any element of MASK scattered to that position is true.

**Class.** Transformational function.

**Arguments.**

**MASK** must be of type logical. It must not be scalar.

**BASE** must be of type logical with the same kind type parameter as MASK. It must not be scalar.

**INDX1, ..., INDXn** must be of type integer and conformable with MASK. The number of INDX arguments must be equal to the rank of BASE.

**Result Type, Type Parameter, and Shape.** Same as BASE.

**Result Value.** The element of the result corresponding to the element \( b \) of BASE has the value \( \text{ANY}(\langle a_1, a_2, \ldots, a_m, b/\rangle) \), where \( (a_1, \ldots, a_m) \) are the elements of MASK associated with \( b \) as described in Section 5.4.4.

**Example.** \( \text{ANY\_SCATTER}(\langle /T, F, F, F/F\rangle, \langle/F, F, T/\rangle, \langle/1, 1, 2, 2/\rangle) \) is
\[
\begin{bmatrix}
T & F & T
\end{bmatrix}.
\]

### 5.7.6 ANY\_SUFFIX(MASK, DIM, SEGMENT, EXCLUSIVE)

**Optional Arguments.** DIM, SEGMENT, EXCLUSIVE

**Description.** Computes a reverse, segmented logical OR scan along dimension DIM of MASK.

**Class.** Transformational function.

**Arguments.**

**MASK** must be of type logical. It must not be scalar.
SECTION 5. INTRINSIC AND LIBRARY PROCEDURES

DIM (optional) must be scalar and of type integer with a value in the range \(1 \leq \text{DIM} \leq n\), where \(n\) is the rank of \text{MASK}.

SEGMENT (optional) must be of type logical and must have the same shape as \text{MASK}.

EXCLUSIVE (optional) must be of type logical and must be scalar.

Result Type, Type Parameter, and Shape. Same as \text{MASK}.

Result Value. Element \(r\) of the result has the value \(\text{ANY}(\langle a_1, \ldots, a_m \rangle)\) where \((a_1, \ldots, a_m)\) is the (possibly empty) set of elements of \text{MASK} selected to contribute to \(r\) by the rules stated in Section 5.4.5.

Example. \text{ANY.SUFFIX}(\langle F,T,F,F,F/\rangle, \text{SEGMENT}=\langle F,F,F,T,T/\rangle)\) is \[
\begin{bmatrix}
T & T & F & F & F \\
\end{bmatrix}
\]

5.7.7 COPY.PREFIX(ARRAY, DIM, SEGMENT)

Optional Arguments. DIM, SEGMENT

Description. Computes a segmented copy scan along dimension \(\text{DIM}\) of \text{ARRAY}.

Class. Transformational function.

Arguments.

\text{ARRAY} may be of any type. It must not be scalar.

DIM (optional) must be scalar and of type integer with a value in the range \(1 \leq \text{DIM} \leq n\), where \(n\) is the rank of \text{ARRAY}.

SEGMENT (optional) must be of type logical and must have the same shape as \text{ARRAY}.

Result Type, Type Parameter, and Shape. Same as \text{ARRAY}.

Result Value. Element \(r\) of the result has the value \(a_1\) where \((a_1, \ldots, a_m)\) is the set, in array element order, of elements of \text{ARRAY} selected to contribute to \(r\) by the rules stated in Section 5.4.5.

Example. \text{COPY.PREFIX}(\langle 1,2,3,4,5/\rangle, \text{SEGMENT}=\langle F,F,F,T,T/\rangle)\) is \[
\begin{bmatrix}
1 & 1 & 1 & 4 & 4 \\
\end{bmatrix}
\]

5.7.8 COPY.SCATTER(ARRAY,BASE,INDX1, ..., INDXn, MASK)

Optional Argument. \text{MASK}

Description. Scatters elements of \text{ARRAY} selected by \text{MASK} to positions of the result indicated by index arrays \text{INDX1}, ..., \text{INDXn}. Each element of the result is equal to one of the elements of \text{ARRAY} scattered to that position or, if there is none, to the corresponding element of \text{BASE}.
5.7. SPECIFICATIONS OF LIBRARY PROCEDURES

Class. Transformational function.

Arguments.

ARRAY may be of any type. It must not be scalar.

BASE must be of the same type and kind type parameter as

ARRAY.

INDX1, ..., INDXn must be of type integer and conformable with ARRAY. The

number of INDX arguments must be equal to the rank of

BASE.

MASK (optional) must be of type logical and must be conformable with

ARRAY.

Result Type, Type Parameter, and Shape. Same as BASE.

Result Value. Let $S$ be the set of elements of ARRAY associated with element $b$ of

BASE as described in Section 5.4.4.

If $S$ is empty, then the element of the result corresponding to the element $b$ of BASE

has the same value as $b$.

If $S$ is non-empty, then the element of the result corresponding to the element $b$ of

BASE is the result of choosing one element from $S$. HPF does not specify how the

choice is to be made; the mechanism is processor dependent.

Example. COPY_SCATTER( (/1, 2, 3, 4/), (/7, 8, 9/), (/1, 1, 2, 2/)) is

$[x, y, 9]$, where $x$ is a member of the set $\{1,2\}$ and $y$ is a member of the set

$\{3,4\}$.

5.7.9 COPY_SUFFIX(ARRAY, DIM, SEGMENT)

Optional Arguments. DIM, SEGMENT

Description. Computes a reverse, segmented copy scan along dimension DIM of

ARRAY.

Class. Transformational function.

Arguments.

ARRAY may be of any type. It must not be scalar.

DIM (optional) must be scalar and of type integer with a value in the

range $1 \leq \text{DIM} \leq n$, where $n$ is the rank of ARRAY.

SEGMENT (optional) must be of type logical and must have the same shape as

ARRAY.

Result Type, Type Parameter, and Shape. Same as ARRAY.

Result Value. Element $r$ of the result has the value $a_m$ where $(a_1, ..., a_m)$ is the

set, in array element order, of elements of ARRAY selected to contribute to $r$ by the

rules stated in Section 5.4.5.

Example. COPY_SUFFIX( (/1,2,3,4,5/), SEGMENT= (/F,F,F,T,T/) ) is

$$[3 \ 3 \ 3 \ 5 \ 5]$$.
5.7.10 COUNT_PREFIX(MASK, DIM, SEGMENT, EXCLUSIVE)

Optional Arguments. DIM, SEGMENT, EXCLUSIVE

Description. Computes a segmented COUNT scan along dimension DIM of MASK.

Class. Transformational function.

Arguments.

MASK must be of type logical. It must not be scalar.

DIM (optional) must be scalar and of type integer with a value in the range \(1 \leq \text{DIM} \leq n\), where \(n\) is the rank of MASK.

SEGMENT (optional) must be of type logical and must have the same shape as MASK.

EXCLUSIVE (optional) must be of type logical and must be scalar.

Result Type, Type Parameter, and Shape. The result is of type default integer and of the same shape as MASK.

Result Value. Element \(r\) of the result has the value \(\text{COUNT}((/ a_1, \ldots, a_m /))\) where \((a_1, \ldots, a_m)\) is the (possibly empty) set of elements of MASK selected to contribute to \(r\) by the rules stated in Section 5.4.5.

Example. \(\text{COUNT}_\text{PREFIX}( (/F,T,T,T,T/), \text{SEGMENT} = (/F,F,F,T,T/) )\) is \([ 0 \ 1 \ 2 \ 1 \ 2 ]\).

5.7.11 COUNT_SCATTER(MASK,BASE,INDX1, ..., INDXn)

Description. Scatters elements of MASK to positions of the result indicated by index arrays INDX1, ..., INDXn. Each element of the result is the sum of the corresponding element of BASE and the number of true elements of MASK scattered to that position.

Class. Transformational function.

Arguments.

MASK must be of type logical. It must not be scalar.

BASE must be of type integer. It must not be scalar.

INDX1, ..., INDXn must be of type integer and conformable with MASK. The number of INDX arguments must be equal to the rank of BASE.

Result Type, Type Parameter, and Shape. Same as BASE.

Result Value. The element of the result corresponding to the element \(b\) of BASE has the value \(b + \text{COUNT}((/a_1, a_2, ..., a_m/))\), where \((a_1, \ldots, a_m)\) are the elements of MASK associated with \(b\) as described in Section 5.4.4.

Example. \(\text{COUNT}_\text{SCATTER}((/T, T, T, F/), (/1, -1, 0/), (/1, 1, 2, 2/) )\) is \([ 3 \ 0 \ 0 ]\).
5.7.12 COUNT_SUFFIX(MASK, DIM, SEGMENT, EXCLUSIVE)

Optional Arguments. DIM, SEGMENT, EXCLUSIVE

Description. Computes a reverse, segmented COUNT scan along dimension DIM of MASK.

Class. Transformational function.

Arguments.

MASK must be of type logical. It must not be scalar.

DIM (optional) must be scalar and of type integer with a value in the range 1 ≤ DIM ≤ n, where n is the rank of MASK.

SEGMENT (optional) must be of type logical and must have the same shape as MASK.

EXCLUSIVE (optional) must be of type logical and must be scalar.

Result Type, Type Parameter, and Shape. The result is of type default integer and of the same shape as MASK.

Result Value. Element r of the result has the value COUNT(/ a1, ..., am /) where (a1, ..., am) is the (possibly empty) set of elements of MASK selected to contribute to r by the rules stated in Section 5.4.5.

Example. COUNT_SUFFIX((/T,F,T,T,T/), SEGMENT=(/F,F,F,T,T/)) is

[2 1 1 2 1].

5.7.13 GRADE_DOWN(ARRAY,DIM)

Optional Argument. DIM

Description. Produces a permutation of the indices of an array, sorted by descending array element values.

Class. Transformational function.

Arguments.

ARRAY must be of type integer, real, or character.

DIM (optional) must be scalar and of type integer with a value in the range 1 ≤ DIM ≤ n, where n is the rank of ARRAY. The corresponding actual argument must not be an optional dummy argument.

Result Type, Type Parameter, and Shape. The result is of type default integer. If DIM is present, the result has the same shape as ARRAY. If DIM is absent, the result has shape (/ SIZE(SHAPE(ARRAY)), PRODUCT(SHAPE(ARRAY)) /).
Result Value.

Case (i): The result of \( S = \text{GRADE\_DOWN}(\text{ARRAY}) \) has the property that if one computes the rank-one array \( B \) of size \( \text{PRODUCT} (\text{SHAPE}(\text{ARRAY})) \) by
\[
\text{FORALL} \ (K=1: \text{SIZE}(B,1)) \ B(K) = \text{ARRAY}(S(1,K), S(2,K), \ldots, S(N,K))
\]
where \( N \) has the value \( \text{SIZE}(\text{SHAPE}(\text{ARRAY})) \), then \( B \) is sorted in descending order; moreover, all of the columns of \( S \) are distinct, that is, if \( j \neq m \) then \( \text{ALL}(S(:,j) \ \text{EQ} \ \text{S}(:,m)) \ will be false. \) The sort is stable; if \( j \leq m \) and \( B(j) = B(m) \), then \( \text{ARRAY}(S(1,j), S(2,j), \ldots, S(n,j)) \ precedes \( \text{ARRAY}(S(1,m), S(2,m), \ldots, S(n,m)) \) in the array element ordering of \( \text{ARRAY} \).

Case (ii): The result of \( R = \text{GRADE\_DOWN}(\text{ARRAY}, \text{DIM}=K) \) has the property that if one computes the array \( B(i_1, i_2, \ldots, i_k, \ldots, i_n) = \text{ARRAY}(i_1, i_2, \ldots, R(i_1, i_2, \ldots, i_k, \ldots, i_n), \ldots, i_n) \) then for all \( i_1, i_2, \ldots, (\text{omit } i_k), \ldots, i_n, \) the vector \( B(i_1, i_2, \ldots, \ldots, i_n) \) is sorted in descending order; moreover, \( R(i_1, i_2, \ldots, \ldots, i_n) \) is a permutation of all the integers in the range
\( \text{LBBOUND}(\text{ARRAY}, K) : \text{UBBOUND}(\text{ARRAY}, K) \). The sort is stable; that is, if \( j \leq m \) and \( B(i_1, i_2, \ldots, j, \ldots, i_n) = B(i_1, i_2, \ldots, m, \ldots, i_n) \), then
\( R(i_1, i_2, \ldots, j, \ldots, i_n) \leq R(i_1, i_2, \ldots, m, \ldots, i_n) \).

Examples.

Case (i): \( \text{GRADE\_DOWN}(\ (/30, 20, 30, 40, -10/) ) \) is a rank two array of shape
\[
\begin{bmatrix}
1 & 5 \\
4 & 1 & 3 & 2 & 5
\end{bmatrix}
\]
with the value \( \begin{bmatrix} 1 & 2 & 3 & 3 & 1 & 2 & 1 & 3 \\
2 & 1 & 3 & 2 & 3 & 3 & 1 & 1 \end{bmatrix} \). (To produce a rank-one result, the optional \( \text{DIM} = 1 \) argument must be used.)

If \( A \) is the array \( \begin{bmatrix} 1 & 9 & 2 \\
4 & 5 & 2 \\
1 & 2 & 4 \end{bmatrix} \),
then \( \text{GRADE\_DOWN}(A) \) has the value \( \begin{bmatrix} 1 & 2 & 2 & 3 & 3 & 1 & 2 & 1 & 3 \\
2 & 1 & 3 & 2 & 3 & 3 & 1 & 1 \end{bmatrix} \).

Case (ii): If \( A \) is the array \( \begin{bmatrix} 1 & 9 & 2 \\
4 & 5 & 2 \\
1 & 2 & 4 \end{bmatrix} \),
then \( \text{GRADE\_DOWN}(A, \text{DIM} = 1) \) has the value \( \begin{bmatrix} 2 & 1 & 3 \\
1 & 2 & 1 \\
3 & 3 & 2 \end{bmatrix} \).

5.7.14 \( \text{GRADE\_UP}(\text{ARRAY}, \text{DIM}) \)

Optional Argument. \( \text{DIM} \)

Description. Produces a permutation of the indices of an array, sorted by ascending array element values.

Class. Transformational function.

Arguments.
ARRAY must be of type integer, real, or character.

DIM (optional) must be scalar and of type integer with a value in the range \( 1 \leq \text{DIM} \leq n \), where \( n \) is the rank of ARRAY. The corresponding actual argument must not be an optional dummy argument.

**Result Type, Type Parameter, and Shape.** The result is of type default integer. If DIM is present, the result has the same shape as ARRAY. If DIM is absent, the result has shape \(/ \text{ SIZE}(\text{SHAPE}(\text{ARRAY})), \text{PRODUCT}(\text{SHAPE}(\text{ARRAY})) /\).**

**Result Value.**

**Case (i):** The result of \( S = \text{GRADE}_\text{UP}(\text{ARRAY}) \) has the property that if one computes the rank-one array \( B \) of size \( \text{PRODUCT}(\text{SHAPE}(\text{ARRAY})) \) by

\[
\text{FORALL } (K=1: \text{SIZE}(B,1)) \quad B(K) = \text{ARRAY}(S(1,K), S(2,K), \ldots, S(N,K))
\]

where \( N \) has the value \( \text{SIZE}(\text{SHAPE}(\text{ARRAY})) \), then \( B \) is sorted in ascending order; moreover, all of the columns of \( S \) are distinct, that is, if \( j \neq m \) then \( \text{ALL}(S(:, j)) \) will be false. The sort is stable; if \( j \leq m \) and \( B(j) = B(m) \), then \( \text{ARRAY}(S(1,j), S(2,j), \ldots, S(n,j)) \) precedes \( \text{ARRAY}(S(1,m), S(2,m), \ldots, S(n,m)) \) in the array element ordering of ARRAY.

**Case (ii):** The result of \( R = \text{GRADE}_\text{UP}(\text{ARRAY}, \text{DIM}=K) \) has the property that if one computes the array \( B(i_1, i_2, \ldots, i_k, \ldots, i_n) = \text{ARRAY}(i_1, i_2, \ldots, R(i_1, i_2, \ldots, i_k, \ldots, i_n), \ldots, i_n) \) then for all \( i_1, i_2, \ldots, (\text{omit } i_k), \ldots, i_n \), the vector \( B(i_1, i_2, \ldots, i_n) \) is sorted in ascending order; moreover, \( R(i_1, i_2, \ldots, i_n) \) is a permutation of all the integers in the range \( \text{LBOUND}(\text{ARRAY}, K):\text{UBOUND}(\text{ARRAY}, K) \). The sort is stable; that is, if \( j \leq m \) and \( B(i_1, i_2, \ldots, j, \ldots, i_n) = B(i_1, i_2, \ldots, m, \ldots, i_n) \), then \( R(i_1, i_2, \ldots, j, \ldots, i_n) \leq R(i_1, i_2, \ldots, m, \ldots, i_n) \).

**Examples.**

**Case (i):** \( \text{GRADE}_\text{UP}(\sqrt{30}, 20, 30, 40, -10/) \) is a rank two array of shape \[
\begin{bmatrix}
1 & 5 \\
\end{bmatrix}
\] with the value \[
\begin{bmatrix}
5 & 2 & 1 & 3 & 4 \\
\end{bmatrix}.
\] (To produce a rank-one result, the optional \( \text{DIM} = 1 \) argument must be used.)

If \( A \) is the array \[
\begin{bmatrix}
1 & 9 & 2 \\
4 & 5 & 2 \\
1 & 2 & 4
\end{bmatrix} ,
\]
then \( \text{GRADE}_\text{UP}(A) \) has the value \[
\begin{bmatrix}
1 & 3 & 3 & 1 & 2 & 2 & 3 & 2 & 1 \\
1 & 1 & 2 & 3 & 3 & 1 & 3 & 2 & 2
\end{bmatrix} .
\]

**Case (ii):** If \( A \) is the array \[
\begin{bmatrix}
1 & 9 & 2 \\
4 & 5 & 2 \\
1 & 2 & 4
\end{bmatrix} ,
\]
then \( \text{GRADE}_\text{UP}(A, \text{DIM} = 1) \) has the value \[
\begin{bmatrix}
1 & 3 & 1 \\
3 & 2 & 2 \\
2 & 1 & 3
\end{bmatrix} .
\]
5.7.15 HPF_ALIGNMENT(ALIGNEE, LB, UB, STRIDE, AXIS_MAP, IDENTITY_MAP, DYNAMIC, NCOPIES)

**Optional Arguments.** LB, UB, STRIDE, AXIS_MAP, IDENTITY_MAP, DYNAMIC, NCOPIES

**Description.** Returns information regarding the correspondence of a variable and the *align-target* (array or template) to which it is ultimately aligned.

**Class.** Mapping inquiry subroutine.

**Arguments.**

**ALIGNEE** may be of any type. It may be scalar or array valued. It must not be an assumed-size array. It must not be a structure component. If it is a member of an aggregate variable group, then it must be an aggregate cover of the group. (See Section 7 for the definitions of “aggregate variable group” and “aggregate cover.”) It must not be a pointer that is disassociated or an allocatable array that is not allocated. It is an INTENT (IN) argument.

If ALIGNEE is a pointer, information about the alignment of its target is returned. The target must not be an assumed-size dummy argument or a section of an assumed-size dummy argument. If the target is (a section of) a member of an aggregate variable group, then the member must be an aggregate cover of the group. The target must not be a structure component, but the pointer may be.

**LB (optional)** must be of type default integer and of rank one. Its size must be at least equal to the rank of ALIGNEE. It is an INTENT (OUT) argument. The first element of the *i*th axis of ALIGNEE is ultimately aligned to the LB(*i*)th *align-target* element along the axis of the *align-target* associated with the *i*th axis of ALIGNEE. If the *i*th axis of ALIGNEE is a collapsed axis, LB(*i*) is processor dependent.

**UB (optional)** must be of type default integer and of rank one. Its size must be at least equal to the rank of ALIGNEE. It is an INTENT (OUT) argument. The last element of the *i*th axis of ALIGNEE is ultimately aligned to the UB(*i*)th *align-target* element along the axis of the *align-target* associated with the *i*th axis of ALIGNEE. If the *i*th axis of ALIGNEE is a collapsed axis, UB(*i*) is processor dependent.

**STRIDE (optional)** must be of type default integer and of rank one. Its size must be at least equal to the rank of ALIGNEE. It is an INTENT (OUT) argument. The *i*th element of STRIDE is set to the stride used in aligning the elements of ALIGNEE along its *i*th axis. If the *i*th axis of ALIGNEE is a collapsed axis, STRIDE(*i*) is zero.
AXIS_MAP (optional) must be of type default integer and of rank one. Its size must be at least equal to the rank of ALIGNEE. It is an INTENT (OUT) argument. The $i^{th}$ element of AXIS_MAP is set to the align-target axis associated with the $i^{th}$ axis of ALIGNEE. If the $i^{th}$ axis of ALIGNEE is a collapsed axis, AXIS_MAP($i$) is 0.

IDENTITY_MAP (optional) must be scalar and of type default logical. It is an INTENT (OUT) argument. It is set to true if the ultimate align-target associated with ALIGNEE has a shape identical to ALIGNEE, the axes are mapped using the identity permutation, and the strides are all positive (and therefore equal to 1, because of the shape constraint); otherwise it is set to false. If a variable has not appeared as an alignee in an ALIGN or REALIGN directive, and does not have the INHERIT attribute, then IDENTITY_MAP must be true; it can be true in other circumstances as well.

DYNAMIC (optional) must be scalar and of type default logical. It is an INTENT (OUT) argument. It is set to true if ALIGNEE has the DYNAMIC attribute; otherwise it is set to false. If ALIGNEE has the pointer attribute, then the result applies to ALIGNEE itself rather than its target.

NCOPIES (optional) must be scalar and of type default integer. It is an INTENT (OUT) argument. It is set to the number of copies of ALIGNEE that are ultimately aligned to align-target. For a non-replicated variable, it is set to one.

Examples. If ALIGNEE is scalar, then no elements of LB, UB, STRIDE, or AXIS_MAP are set.

Given the declarations

```fortran
REAL PI = 3.1415927
POINTER P_TO_A(:)
DIMENSION A(10,10),B(20,30),C(20,40,10),D(40)
!HPFS TEMPLATE T(40,20)
!HPFS DYNAMIC A
!HPFS ALIGN A(1,:) WITH T(1+3*I,2:20:2)
!HPFS ALIGN C(1,:,J) WITH T(J,21-I)
!HPFS ALIGN D(I) WITH T(I,4)
!HPFS PROCESSORS PROCS(4,2), SCALARPROC
!HPFS DISTRIBUTE T(BLOCK,BLOCK) ONTO PROCS
!HPFS DISTRIBUTE B(CYCLIC,BLOCK) ONTO PROCS
!HPFS DISTRIBUTE ONTO SCALARPROC :: PI
   P_TO_A => A(3:9:2, 6)
```

assuming that the actual mappings are as the directives specify, the results of HPF_ALIGNMENT are:
### Section 5. Intrinsic and Library Procedures

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>P_TO_A</th>
</tr>
</thead>
<tbody>
<tr>
<td>LB</td>
<td>4, 2</td>
<td>1, 1</td>
<td>[1, N/A, 1]</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>UB</td>
<td>31, 20</td>
<td>20, 30</td>
<td>20, N/A, 10</td>
<td>40</td>
<td>28</td>
</tr>
<tr>
<td>STRIDE</td>
<td>3, 2</td>
<td>1, 1</td>
<td>-1, 0, 1</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>AXIS_MAP</td>
<td>1, 2</td>
<td>1, 2</td>
<td>[2, 0, 1]</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>IDENTITY_MAP</td>
<td>false</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>DYNAMIC</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>NCOPIES</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

where “N/A” denotes a processor-dependent result. To illustrate the use of NCOPIES, consider:

```fortran
LOGICAL BOZO(20, 20), RONALD_MCDONALD(20)
!HPFS$ TEMPLATE EMMETT_KELLY(100, 100)
!HPFS$ ALIGN RONALD_MCDONALD(I) WITH BOZO(I, *)
!HPFS$ ALIGN BOZO(J, K) WITH EMMETT_KELLY(J, 5*K)

CALL HPF_ALIGNMENT(RONALD_MCDONALD, NCOPIES = NC) sets NC to 20. Now consider:

LOGICAL BOZO(20, 20), RONALD_MCDONALD(20)
!HPFS$ TEMPLATE WILLIE_WHISTLE(100)
!HPFS$ ALIGN RONALD_MCDONALD(I) WITH BOZO(I, *)
!HPFS$ ALIGN BOZO(J, *) WITH WILLIE_WHISTLE(5*J)

CALL HPF_ALIGNMENT(RONALD_MCDONALD, NCOPIES = NC) sets NC to one.
```

#### 5.7.16 HPF_TEMPLATE(ALIGNTEE, TEMPLATE_RANK, LB, UB, AXIS_TYPE, AXIS_INFO, NUMBER_ALIGNED, DYNAMIC)

**Optional Arguments.** LB, UB, AXIS_TYPE, AXIS_INFO, NUMBER_ALIGNED, TEMPLATE_RANK, DYNAMIC

**Description.** The HPF_TEMPLATE subroutine returns information regarding the ultimate align-target associated with a variable; HPF_TEMPLATE returns information concerning the variable from the template’s point of view (assuming the alignment is to a template rather than to an array), while HPF_ALIGNMENT returns information from the variable’s point of view.

**Class.** Mapping inquiry subroutine.

**Arguments.**

- **ALIGNTEE**
  - May be of any type. It may be scalar or array valued. It must not be an assumed-size array. It must not be a structure component. If it is a member of an aggregate variable group, then it must be an aggregate cover of the group. (See Section 7 for the definitions of “aggregate variable group” and “aggregate cover.”) It must not be a pointer that is disassociated or an allocatable array that is not allocated. It is an **INTENT (IN)** argument.
  - If ALIGNTEE is a pointer, information about the alignment of its target is returned. The target must not be
an assumed-size dummy argument or a section of an assumed-size dummy argument. If the target is (a section of) a member of an aggregate variable group, then the member must be an aggregate cover of the group. The target must not be a structure component, but the pointer may be.

`TEMPLATE_RANK` (optional) must be scalar and of type default integer. It is an `INTENT` (OUT) argument. It is set to the rank of the ultimate `align-target`. This can be different from the rank of the `ALIGNEE`, due to collapsing and replicating.

`LB` (optional) must be of type default integer and of rank one. Its size must be at least equal to the rank of the `align-target` to which `ALIGNEE` is ultimately aligned; this is the value returned in `TEMPLATE_RANK`. It is an `INTENT` (OUT) argument. The i\textsuperscript{th} element of `LB` contains the declared `align-target` lower bound for the i\textsuperscript{th} template axis.

`UB` (optional) must be of type default integer and of rank one. Its size must be at least equal to the rank of the `align-target` to which `ALIGNEE` is ultimately aligned; this is the value returned in `TEMPLATE_RANK`. It is an `INTENT` (OUT) argument. The i\textsuperscript{th} element of `UB` contains the declared `align-target` upper bound for the i\textsuperscript{th} template axis.

`AXIS_TYPE` (optional) must be a rank one array of type default character. It may be of any length, although it must be of length at least 10 in order to contain the complete value. Its elements are set to the values below as if by a character intrinsic assignment statement. Its size must be at least equal to the rank of the `align-target` to which `ALIGNEE` is ultimately aligned; this is the value returned in `TEMPLATE_RANK`. It is an `INTENT` (OUT) argument. The i\textsuperscript{th} element of `AXIS_TYPE` contains information about the i\textsuperscript{th} axis of the `align-target`. The following values are defined by HPF (implementations may define other values):

'NORMAL' The `align-target` axis has an axis of `ALIGNEE` aligned to it. For elements of `AXIS_TYPE` assigned this value, the corresponding element of `AXIS_INFO` is set to the number of the axis of `ALIGNEE` aligned to this `align-target` axis.

'REPLICATED' `ALIGNEE` is replicated along this `align-target` axis. For elements of `AXIS_TYPE` assigned this value, the corresponding element of `AXIS_INFO` is set to the number of copies of `ALIGNEE` along this `align-target` axis.

'SINGLE' `ALIGNEE` is aligned with one coordinate of the `align-target` axis. For elements of `AXIS_TYPE` assigned
this value, the corresponding element of \texttt{AXIS_INFO} is set to the \textit{align-target} coordinate to which \texttt{ALIGNEE} is aligned.

\texttt{AXIS_INFO} (optional) must be of type default integer and of rank one. Its size must be at least equal to the rank of the \textit{align-target} to which \texttt{ALIGNEE} is ultimately aligned; this is the value returned in \texttt{TEMPLATE_RANK}. It is an \texttt{INTENT} (OUT) argument. See the description of \texttt{AXIS_TYPE} above.

\texttt{NUMBER_ALIGNED} (optional) must be scalar and of type default integer. It is an \texttt{INTENT} (OUT) argument. It is set to the total number of variables aligned to the ultimate \textit{align-target}. This is the number of variables that are moved if the \textit{align-target} is redistributed.

\texttt{DYNAMIC} (optional) must be scalar and of type default logical. It is an \texttt{INTENT} (OUT) argument. It is set to true if the \textit{align-target} has the \texttt{DYNAMIC} attribute, and to false otherwise.

\textbf{Example.} Given the declarations in the example of Section 5.7.15, and assuming that the actual mappings are as the directives specify, the results of \texttt{HPF TEMPLATE} are:

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>LB</td>
<td>[1, 1]</td>
<td>[1, 1]</td>
<td>[1, 1]</td>
</tr>
<tr>
<td>UB</td>
<td>[40, 20]</td>
<td>[40, 20]</td>
<td>[40, 20]</td>
</tr>
<tr>
<td>\texttt{AXIS_TYPE}</td>
<td>[&quot;NORMAL&quot;, &quot;NORMAL&quot;]</td>
<td>[&quot;NORMAL&quot;, &quot;NORMAL&quot;]</td>
<td>[&quot;NORMAL&quot;, &quot;SINGLE&quot;]</td>
</tr>
<tr>
<td>\texttt{AXIS_INFO}</td>
<td>[1, 2]</td>
<td>[3, 1]</td>
<td>[1, 4]</td>
</tr>
<tr>
<td>\texttt{NUMBER_ALIGNED}</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>\texttt{TEMPLATE_RANK}</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>\texttt{DYNAMIC}</td>
<td>false</td>
<td>false</td>
<td>false</td>
</tr>
</tbody>
</table>

5.7.17 \texttt{HPF DISTRIBUTION(DISTRIBUTEE, AXIS_TYPE, AXIS_INFO, PROCESSORS_RANK, PROCESSORS SHAPE)}

\textbf{Optional Arguments.} \texttt{AXIS_TYPE, AXIS_INFO, PROCESSORS_RANK, PROCESSORS SHAPE}

\textbf{Description.} The \texttt{HPF DISTRIBUTION} subroutine returns information regarding the distribution of the ultimate \textit{align-target} associated with a variable.

\textbf{Class.} Mapping inquiry subroutine.

\textbf{Arguments.}

\texttt{DISTRIBUTEE} may be of any type. It may be scalar or array valued. It must not be an assumed-size array. It must not be a structure component. If it is a member of an aggregate variable group, then it must be an aggregate cover of the group. (See Section 7 for the definitions of “aggregate
variable group" and "aggregate cover.") It must not be a pointer that is disassociated or an allocatable array that is not allocated. It is an INTENT (IN) argument.

If DISTRIBUTEE is a pointer, information about the distribution of its target is returned. The target must not be an assumed-size dummy argument or a section of an assumed-size dummy argument. If the target is (a section of) a member of an aggregate variable group, then the member must be an aggregate cover of the group. The target must not be a structure component, but the pointer may be.

**AXIS.TYPE (optional)** must be a rank one array of type default character. It may be of any length, although it must be of length at least 9 in order to contain the complete value. Its elements are set to the values below as if by a character intrinsic assignment statement. Its size must be at least equal to the rank of the align-target to which DISTRIBUTEE is ultimately aligned; this is the value returned by HPF TEMPLATE in TEMPLATE_RANK). It is an INTENT (OUT) argument. Its i-th element contains information on the distribution of the i-th axis of that align-target. The following values are defined by HPF (implementations may define other values):

- **'BLOCK'** The axis is distributed BLOCK. The corresponding element of AXIS_INFO contains the block size.
- **'COLLAPSED'** The axis is collapsed (distributed with the "*" specification). The value of the corresponding element of AXIS_INFO is processor dependent.
- **'CYCLIC'** The axis is distributed CYCLIC. The corresponding element of AXIS_INFO contains the block size.

**AXIS_INFO (optional)** must be a rank one array of type default integer, and size at least equal to the rank of the align-target to which DISTRIBUTEE is ultimately aligned (which is returned by HPF TEMPLATE in TEMPLATE_RANK). It is an INTENT (OUT) argument. The i-th element of AXIS_INFO contains the block size in the block or cyclic distribution of the i-th axis of the ultimate align-target of DISTRIBUTEE; if that axis is a collapsed axis, then the value is processor dependent.

**PROCCESSORS_RANK (optional)** must be scalar and of type default integer. It is set to the rank of the processor arrangement onto which DISTRIBUTEE is distributed. It is an INTENT (OUT) argument.

**PROCCESSORS_SHAPE (optional)** must be a rank one array of type default integer and of size at least equal to the value, m, returned in PROCESSORS_RANK. It is an INTENT (OUT) argument. Its first m
elements are set to the shape of the processor arrangement onto which DISTRIBUTEE is mapped. (It may be necessary to call HPF.DISTRIBUTION twice, the first time to obtain the value of PROCESSORS.RANK in order to allocate PROCESSORS.SHAPE.)

Example. Given the declarations in the example of Section 5.7.15, and assuming that the actual mappings are as the directives specify, the results of HPF.DISTRIBUTION are:

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>PI</th>
</tr>
</thead>
<tbody>
<tr>
<td>AXIS.TYPE</td>
<td>['BLOCK', 'BLOCK']</td>
<td>['CYCLIC', 'BLOCK']</td>
<td></td>
</tr>
<tr>
<td>AXIS.INFO</td>
<td>[10, 10]</td>
<td>[1, 15]</td>
<td></td>
</tr>
<tr>
<td>PROCESSORS.SHAPE</td>
<td>[4, 2]</td>
<td>[4, 2]</td>
<td></td>
</tr>
<tr>
<td>PROCESSORS.RANK</td>
<td>2</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>

5.7.18 IALL(ARRAY, DIM, MASK)

Optional Arguments. DIM, MASK

Description. Computes a bitwise logical AND reduction along dimension DIM of ARRAY.

Class. Transformational function.

Arguments.

ARRAY must be of type integer. It must not be scalar.

DIM (optional) must be scalar and of type integer with a value in the range $1 \leq DIM \leq n$, where $n$ is the rank of ARRAY. The corresponding actual argument must not be an optional dummy argument.

MASK (optional) must be of type logical and must be conformable with ARRAY.

Result Type, Type Parameter, and Shape. The result is of type integer with the same kind type parameter as ARRAY. It is scalar if DIM is absent or if ARRAY has rank one; otherwise, the result is an array of rank $n - 1$ and shape $(d_1, d_2, \ldots, d_{DIM-1}, d_{DIM+1}, \ldots, d_n)$ where $(d_1, d_2, \ldots, d_n)$ is the shape of ARRAY.

Result Value.

Case (i): The result of IALL(ARRAY) is the IAND reduction of all the elements of ARRAY. If ARRAY has size zero, the result is equal to a processor-dependent integer value $x$ with the property that IAND(I, x) = I for all integers I of the same kind type parameter as ARRAY. See Section 5.4.3.

Case (ii): The result of IALL(ARRAY, MASK=MASK) is the IAND reduction of all the elements of ARRAY corresponding to the true elements of MASK; if MASK contains no true elements, the result is equal to a processor-dependent integer value $x$ (of the same kind type parameter as ARRAY) with the property that IAND(I, x) = I for all integers I.
Case (iii): If ARRAY has rank one, IALL(ARRAY, DIM=1 [,MASK]) has a value equal to that of IALL(ARRAY [,MASK]). Otherwise, the value of element $(s_1, s_2, \ldots, s_{\text{DIM}-1}, s_{\text{DIM}+1}, \ldots, s_n)$ of IALL(ARRAY, DIM=1 [,MASK]) is equal to IALL(ARRAY$(s_1, s_2, \ldots, s_{\text{DIM}-1}, s_{\text{DIM}+1}, \ldots, s_n)$ [,MASK = MASK$(s_1, s_2, \ldots, s_{\text{DIM}-1}, s_{\text{DIM}+1}, \ldots, s_n)$])

Examples.

Case (i): The value of IALL( (/7, 6, 3, 2/)) is 2.

Case (ii): The value of IALL(C, MASK = BTEST(C,0)) is the IAND reduction of the odd elements of C.

Case (iii): If B is the array \[ \begin{bmatrix} 2 & 3 & 5 \\ 3 & 7 & 7 \end{bmatrix} \], then IALL(B, DIM = 1) is \[ \begin{bmatrix} 2 & 3 & 5 \end{bmatrix} \] and IALL(B, DIM = 2) is \[ \begin{bmatrix} 0 & 3 \end{bmatrix} \].

5.7.19 IALL_PREFIX(ARRAY, DIM, MASK, SEGMENT, EXCLUSIVE)

Optional Arguments. DIM, MASK, SEGMENT, EXCLUSIVE

Description. Computes a segmented bitwise logical AND scan along dimension DIM of ARRAY.

Class. Transformational function.

Arguments.

ARRAY must be of type integer. It must not be scalar.

DIM (optional) must be scalar and of type integer with a value in the range $1 \leq \text{DIM} \leq n$, where $n$ is the rank of ARRAY.

MASK (optional) must be of type logical and must be conformable with ARRAY.

SEGMENT (optional) must be of type logical and must have the same shape as ARRAY.

EXCLUSIVE (optional) must be of type logical and must be scalar.

Result Type, Type Parameter, and Shape. Same as ARRAY.

Result Value. Element $r$ of the result has the value IALL((/ $a_1, \ldots, a_m$ /)) where $(a_1, \ldots, a_m)$ is the (possibly empty) set of elements of ARRAY selected to contribute to $r$ by the rules stated in Section 5.4.5.

Example. IALL_PREFIX( (/1,3,2,4,5/), SEGMENT= (/F,F,F,T,T/) ) is \[ \begin{bmatrix} 1 & 1 & 0 & 4 & 4 \end{bmatrix} \].
5.7.20  **IALL_SCATTER(ARRAY, BASE, INDX1, ..., INDXn, MASK)**

**Optional Argument. MASK**

**Description.** Scatters elements of **ARRAY** selected by **MASK** to positions of the result indicated by index arrays **INDX1**, ..., **INDXn**. The \( j^{th} \) bit of an element of the result is 1 if and only if the \( j^{th} \) bits of the corresponding element of **BASE** and of the elements of **ARRAY** scattered to that position are all equal to 1.

**Class.** Transformational function.

**Arguments.**

**ARRAY** must be of type integer. It must not be scalar.

**BASE** must be of type integer with the same kind type parameter as **ARRAY**. It must not be scalar.

**INDX1, ..., INDXn** must be of type integer and conformable with **ARRAY**. The number of **INDX** arguments must be equal to the rank of **BASE**.

**MASK (optional)** must be of type logical and must be conformable with **ARRAY**.

**Result Type, Type Parameter, and Shape.** Same as **BASE**.

**Result Value.** The element of the result corresponding to the element \( b \) of **BASE** has the value \( \text{IALL}( (/a_1, a_2, ..., a_m, b/) ) \), where \( (a_1, ..., a_m) \) are the elements of **ARRAY** associated with \( b \) as described in Section 5.4.4.

**Example.** \[ \begin{array}{c}
0 \\
2 \\
7
\end{array} \] \[
\text{IALL\_SCATTER}((/1, 2, 3, 6/), (/1, 3, 7/), (/1, 1, 2, 2/))
\]

5.7.21 **IALL\_SUFFIX(ARRAY, DIM, MASK, SEGMENT, EXCLUSIVE)**

**Optional Arguments. DIM, MASK, SEGMENT, EXCLUSIVE**

**Description.** Computes a reverse, segmented bitwise logical AND scan along dimension **DIM** of **ARRAY**.

**Class.** Transformational function.

**Arguments.**

**ARRAY** must be of type integer. It must not be scalar.

**DIM (optional)** must be scalar and of type integer with a value in the range \( 1 \leq \text{DIM} \leq n \), where \( n \) is the rank of **ARRAY**.

**MASK (optional)** must be of type logical and must be conformable with **ARRAY**.

**SEGMENT (optional)** must be of type logical and must have the same shape as **ARRAY**.
EXCLUSIVE (optional) must be of type logical and must be scalar.

Result Type, Type Parameter, and Shape. Same as ARRAY.

Result Value. Element $r$ of the result has the value $\text{IALL}(\{a_1, \ldots, a_m\})$ where $(a_1, \ldots, a_m)$ is the (possibly empty) set of elements of ARRAY selected to contribute to $r$ by the rules stated in Section 5.4.5.

Example. $\text{IALL.SUFFIX}(\{1,3,2,4,5\}, \text{SEGMENT}=(F,F,F,T,T))$ is $[0 2 2 4 5]$.

5.7.22 IANY(ARRAY, DIM, MASK)

Optional Arguments. DIM, MASK

Description. Computes a bitwise logical OR reduction along dimension DIM of ARRAY.

Class. Transformational function.

Arguments.

ARRAY must be of type integer. It must not be scalar.

DIM (optional) must be scalar and of type integer with a value in the range $1 \leq \text{DIM} \leq n$, where $n$ is the rank of ARRAY. The corresponding actual argument must not be an optional dummy argument.

MASK (optional) must be of type logical and must be conformable with ARRAY.

Result Type, Type Parameter, and Shape. The result is of type integer with the same kind type parameter as ARRAY. It is scalar if DIM is absent or if ARRAY has rank one; otherwise, the result is an array of rank $n - 1$ and shape $(d_1, d_2, \ldots, d_{\text{DIM}-1}, d_{\text{DIM}+1}, \ldots, d_n)$ where $(d_1, d_2, \ldots, d_n)$ is the shape of ARRAY.

Result Value.

Case (i): The result of IANY(ARRAY) is the IOR reduction of all the elements of ARRAY. If ARRAY has size zero, the result has the value zero. See Section 5.4.3.

Case (ii): The result of IANY(ARRAY, MASK=MASK) is the IOR reduction of all the elements of ARRAY corresponding to the true elements of MASK; if MASK contains no true elements, the result is zero.

Case (iii): If ARRAY has rank one, IANY(ARRAY, DIM=1 [,MASK]) has a value equal to that of IANY(ARRAY [,MASK]). Otherwise, the value of element $(s_1, s_2, \ldots, s_{\text{DIM}-1}, s_{\text{DIM}+1}, \ldots, s_n)$ of IANY(ARRAY, DIM=1 [,MASK]) is equal to IANY(ARRAY$(s_1, s_2, \ldots, s_{\text{DIM}-1}, :s_{\text{DIM}+1}, \ldots, s_n)$ [,MASK = MASK$(s_1, s_2, \ldots, s_{\text{DIM}-1}, :s_{\text{DIM}+1}, \ldots, s_n)$])
Examples.

Case (i): The value of \texttt{IANY((/9, 8, 3, 2/))} is 11.

Case (ii): The value of \texttt{IANY(C, MASK = BTEST(C, 0))} is the IOR reduction of the odd elements of \texttt{C}.

Case (iii): If \texttt{B} is the array \[
\begin{bmatrix}
2 & 3 & 5 \\
0 & 4 & 2
\end{bmatrix},
\]
then \texttt{IANY(B, DIM = 1)} is \[
\begin{bmatrix}
2 & 7 & 7
\end{bmatrix}
\]
and \texttt{IANY(B, DIM = 2)} is \[
\begin{bmatrix}
7 & 6
\end{bmatrix}.
\]

5.7.23 \texttt{IANY\_PREFIX(ARRAY, DIM, MASK, SEGMENT, EXCLUSIVE)}

Optional Arguments. \texttt{DIM, MASK, SEGMENT, EXCLUSIVE}

Description. Computes a segmented bitwise logical OR scan along dimension \texttt{DIM} of \texttt{ARRAY}.

Class. Transformational function.

Arguments.

\texttt{ARRAY} must be of type integer. It must not be scalar.

\texttt{DIM (optional)} must be scalar and of type integer with a value in the range \(1 \leq \texttt{DIM} \leq n\), where \(n\) is the rank of \texttt{ARRAY}.

\texttt{MASK (optional)} must be of type logical and must be conformable with \texttt{ARRAY}.

\texttt{SEGMENT (optional)} must be of type logical and must have the same shape as \texttt{ARRAY}.

\texttt{EXCLUSIVE (optional)} must be of type logical and must be scalar.

Result Type, Type Parameter, and Shape. Same as \texttt{ARRAY}.

Result Value. Element \(r\) of the result has the value \texttt{IANY((/a_1, \ldots, a_m /))} where \((a_1, \ldots, a_m)\) is the (possibly empty) set of elements of \texttt{ARRAY} selected to contribute to \(r\) by the rules stated in Section 5.4.5.

Example. \texttt{IANY\_PREFIX( (/1,2,3,2,5/), SEGMENT= (/F,F,F,T,T/) )} is \[
\begin{bmatrix}
1 & 3 & 3 & 2 & 7
\end{bmatrix}.
\]

5.7.24 \texttt{IANY\_SCATTER(ARRAY, BASE, INDEX1, \ldots, INDEXn, MASK)}

Optional Argument. \texttt{MASK}

Description. Scatters elements of \texttt{ARRAY} selected by \texttt{MASK} to positions of the result indicated by index arrays \texttt{INDEX1}, \ldots, \texttt{INDEXn}. The \(j^{\text{th}}\) bit of an element of the result is 1 if and only if the \(j^{\text{th}}\) bit of the corresponding element of \texttt{BASE} or of any of the elements of \texttt{ARRAY} scattered to that position is equal to 1.
Class. Transformational function.

Arguments.

ARRAY must be of type integer. It must not be scalar.

BASE must be of type integer with the same kind type parameter as ARRAY. It must not be scalar.

INDX1, ..., INDEXn must be of type integer and conformable with ARRAY. The number of INDEX arguments must be equal to the rank of BASE.

MASK (optional) must be of type logical and must be conformable with ARRAY.

Result Type, Type Parameter, and Shape. Same as BASE.

Result Value. The element of the result corresponding to the element b of BASE has the value IANY(/a1, a2, ..., am, b/), where (a1, ..., am) are the elements of ARRAY associated with b as described in Section 5.4.4.

Example. IANY_SCATTER(/1, 2, 3, 6/), (/1, 3, 7/), (/1, 1, 2, 2/) is 
[ 3 7 7 ].

5.7.25 IANY_SUFFIX(ARRAY, DIM, MASK, SEGMENT, EXCLUSIVE)

Optional Arguments. DIM, MASK, SEGMENT, EXCLUSIVE

Description. Computes a reverse, segmented bitwise logical OR scan along dimension DIM of ARRAY.

Class. Transformational function.

Arguments.

ARRAY must be of type integer. It must not be scalar.

DIM (optional) must be scalar and of type integer with a value in the range 1 \leq DIM \leq n, where n is the rank of ARRAY.

MASK (optional) must be of type logical and must be conformable with ARRAY.

SEGMENT (optional) must be of type logical and must have the same shape as ARRAY.

EXCLUSIVE (optional) must be of type logical and must be scalar.

Result Type, Type Parameter, and Shape. Same as ARRAY.

Result Value. Element r of the result has the value IANY((/ a1, ..., am /)) where (a1, ..., am) is the (possibly empty) set of elements of ARRAY selected to contribute to r by the rules stated in Section 5.4.5.

Example. IANY_SUFFIX( /4,2,3,2,5/), SEGMENT= (/F,F,F,T,T/) is
[ 7 3 3 7 5 ].
5.7.26 \textsc{iparity}(\textsc{array}, \textsc{dim}, \textsc{mask})

Optional Arguments. $\textsc{dim}$, $\textsc{mask}$

Description. Computes a bitwise logical exclusive OR reduction along dimension $\textsc{dim}$ of $\textsc{array}$.

Class. Transformational function.

Arguments.

\textsc{array} \quad \text{must be of type integer. It must not be scalar.}

\textsc{dim} (optional) \quad \text{must be scalar and of type integer with a value in the range } 1 \leq \textsc{dim} \leq n, \text{ where } n \text{ is the rank of } \textsc{array}. \text{ The corresponding actual argument must not be an optional dummy argument.}

\textsc{mask} (optional) \quad \text{must be of type logical and must be conformable with } \textsc{array}.

Result Type, Type Parameter, and Shape. The result is of type integer with the same kind type parameter as $\textsc{array}$. It is scalar if $\textsc{dim}$ is absent or if $\textsc{array}$ has rank one; otherwise, the result is an array of rank $n - 1$ and shape $(d_1, d_2, \ldots, d_{\textsc{dim}-1}, d_{\textsc{dim}+1}, \ldots, d_n)$ where $(d_1, d_2, \ldots, d_n)$ is the shape of $\textsc{array}$.

Result Value.

Case (i): The result of \textsc{iparity}(\textsc{array}) is the \textsc{ieor} reduction of all the elements of $\textsc{array}$. If $\textsc{array}$ has size zero, the result has the value zero. See Section 5.4.3.

Case (ii): The result of \textsc{iparity}(\textsc{array}, \textsc{mask}=\textsc{mask}) is the \textsc{ieor} reduction of all the elements of $\textsc{array}$ corresponding to the true elements of $\textsc{mask}$; if $\textsc{mask}$ contains no true elements, the result is zero.

Case (iii): If $\textsc{array}$ has rank one, \textsc{iparity}(\textsc{array}, \textsc{dim}=1 [,\textsc{mask}]) has a value equal to that of \textsc{iparity}(\textsc{array} [,\textsc{mask}]). Otherwise, the value of element $(s_1, s_2, \ldots, s_{\textsc{dim}-1}, s_{\textsc{dim}+1}, \ldots, s_n)$ of \textsc{iparity}(\textsc{array}, \textsc{dim}=1 [,\textsc{mask}]) is equal to \textsc{iparity}(\textsc{array}(s_1, s_2, \ldots, s_{\textsc{dim}-1}, s_{\textsc{dim}+1}, \ldots, s_n) [,\textsc{mask} = \textsc{mask}(s_1, s_2, \ldots, s_{\textsc{dim}-1}, s_{\textsc{dim}+1}, \ldots, s_n)])

Examples.

Case (i): The value of \textsc{iparity}((/13, 8, 3, 2/)) is 4.

Case (ii): The value of \textsc{iparity}(C, \textsc{mask} = \textsc{btest}(C, 0)) is the \textsc{ieor} reduction of the odd elements of C.

Case (iii): If B is the array \[
\begin{bmatrix}
2 & 3 & 7 \\
0 & 4 & 2
\end{bmatrix},
\] then \textsc{iparity}(B, \textsc{dim} = 1) is \[
\begin{bmatrix}
2 & 7 & 5
\end{bmatrix}
\] and \textsc{iparity}(B, \textsc{dim} = 2) is \[
\begin{bmatrix}
6 & 6
\end{bmatrix}.
\]
5.7.27  IPARI T Y-PRE FIX (ARRAY, DIM, MASK, SEGMENT, EXCLUSIVE)

Optional Arguments. DIM, MASK, SEGMENT, EXCLUSIVE

Description. Computes a segmented bitwise logical exclusive OR scan along dimension DIM of ARRAY.

Class. Transformational function.

Arguments.

ARRAY must be of type integer. It must not be scalar.

DIM (optional) must be scalar and of type integer with a value in the range \(1 \leq DIM \leq n\), where \(n\) is the rank of ARRAY.

MASK (optional) must be of type logical and must be conformable with ARRAY.

SEGMENT (optional) must be of type logical and must have the same shape as ARRAY.

EXCLUSIVE (optional) must be of type logical and must be scalar.

Result Type, Type Parameter, and Shape. Same as ARRAY.

Result Value. Element \(r\) of the result has the value \(\text{IPAR ITY}((/ a_1, \ldots, a_m /))\) where \((a_1, \ldots, a_m)\) is the (possibly empty) set of elements of ARRAY selected to contribute to \(r\) by the rules stated in Section 5.4.5.

Example. \(\text{IPAR ITY-PRE FIX}((/1,2,3,4,5/), \text{SEGMENT}= (/F,F,F,T,T/) )\) is \[
\begin{bmatrix}
1 & 3 & 0 & 4 & 1
\end{bmatrix}
\]

5.7.28  IPAR ITY-SCATTER(ARRAY,BASE,INDX1, ..., INDXn, MASK)

Optional Argument. MASK

Description. Scatters elements of ARRAY selected by MASK to positions of the result indicated by index arrays INDX1, ..., INDXn. The \(j^{th}\)bit of an element of the result is 1 if and only if there are an odd number of ones among the \(j^{th}\)bits of the corresponding element of BASE and the elements of ARRAY scattered to that position.

Class. Transformational function.

Arguments.

ARRAY must be of type integer. It must not be scalar.

BASE must be of type integer with the same kind type parameter as ARRAY. It must not be scalar.

INDX1, ..., INDXn must be of type integer and conformable with ARRAY. The number of INDX arguments must be equal to the rank of BASE.
MASK (optional) must be of type logical and must be conformable with ARRAY.

Result Type, Type Parameter, and Shape. Same as BASE.

Result Value. The element of the result corresponding to the element b of BASE has the value IPARITY( (/a₁,a₂,...,aₘ,b/) ), where (a₁,...,aₘ) are the elements of ARRAY associated with b as described in Section 5.4.4.

Example. IPARITY_SSCATTER( (/1,2,3,6/), (/1,3,7/), (/1,1,2,2/) ) is [ 2 6 7 ].

5.7.29 IPARITY_SUFFIX(ARRAY, DIM, MASK, SEGMENT, EXCLUSIVE)

Optional Arguments. DIM, MASK, SEGMENT, EXCLUSIVE

Description. Computes a reverse, segmented bitwise logical exclusive OR scan along dimension DIM of ARRAY.

Class. Transformational function.

Arguments.

ARRAY must be of type integer. It must not be scalar.

DIM (optional) must be scalar and of type integer with a value in the range 1 ≤ DIM ≤ n, where n is the rank of ARRAY.

MASK (optional) must be of type logical and must be conformable with ARRAY.

SEGMENT (optional) must be of type logical and must have the same shape as ARRAY.

EXCLUSIVE (optional) must be of type logical and must be scalar.

Result Type, Type Parameter, and Shape. Same as ARRAY.

Result Value. Element r of the result has the value IPARITY((/ a₁,...,aₘ /)) where (a₁,...,aₘ) is the (possibly empty) set of elements of ARRAY selected to contribute to r by the rules stated in Section 5.4.5.

Example. IPARITY_SUFFIX( (/1,2,3,4,5/), SEGMENT= (/F,F,F,T,T/) ) is [ 0 1 3 1 5 ].

5.7.30 LEADZ(I)

Description. Return the number of leading zeros in an integer.

Class. Elemental function.

Argument. I must be of type integer.
Result Type and Type Parameter. Same as I.

Result Value. The result is a count of the number of leading 0-bits in the integer I. The model for the interpretation of an integer as a sequence of bits is in Section 13.5.7 of the Fortran 90 Standard. LEADZ(0) is BIT_SIZE(I). For nonzero I, if the leftmost one bit of I occurs in position \( k - 1 \) (where the rightmost bit is bit 0) then LEADZ(I) is BIT_SIZE(I) - k.

Examples. LEADZ(3) has the value BIT_SIZE(3) - 2. For scalar I, LEADZ(I) .EQ. MINVAL( (/ (J, J=0, BIT_SIZE(I) ) /), MASK=M ) where M =(/ (BTEST(I,J), J=BIT_SIZE(I)-1, 0, -1), .TRUE. /). A given integer I may produce different results from LEADZ(I), depending on the number of bits in the representation of the integer (BIT_SIZE(I)). That is because LEADZ counts bits from the most significant bit. Compare with ILEN.

5.7.31 MAXVAL_PREFIX(ARRAY, DIM, MASK, SEGMENT, EXCLUSIVE)

Optional Arguments. DIM, MASK, SEGMENT, EXCLUSIVE

Description. Computes a segmented MAXVAL scan along dimension DIM of ARRAY.

Class. Transformational function.

Arguments.

ARRAY must be of type integer or real. It must not be scalar.

DIM (optional) must be scalar and of type integer with a value in the range \( 1 \leq \text{DIM} \leq n \), where \( n \) is the rank of ARRAY.

MASK (optional) must be of type logical and must be conformable with ARRAY.

SEGMENT (optional) must be of type logical and must have the same shape as ARRAY.

EXCLUSIVE (optional) must be of type logical and must be scalar.

Result Type, Type Parameter, and Shape. Same as ARRAY.

Result Value. Element \( r \) of the result has the value MAXVAL(\(/ a_1, \ldots, a_m /\)) where \( (a_1, \ldots, a_m) \) is the (possibly empty) set of elements of ARRAY selected to contribute to \( r \) by the rules stated in Section 5.4.5.

Example. MAXVAL_PREFIX( /3,4,-5,2,5/), SEGMENT= (/F,F,F,T,T/) is 
\[
\begin{bmatrix}
3 & 4 & 4 & 2 & 5
\end{bmatrix}
\]
5.7.32  MAXVAL_SCATTER(ARRAY,BASE,INDX1, ..., INDXn, MASK)

**Optional Argument.** MASK

**Description.** Scatters elements of ARRAY selected by MASK to positions of the result indicated by index arrays INDX1, ..., INDXn. Each element of the result is assigned the maximum value of the corresponding element of BASE and the elements of ARRAY scattered to that position.

**Class.** Transformational function.

**Arguments.**

- **ARRAY** must be of type integer or real. It must not be scalar.
- **BASE** must be of the same type and kind type parameter as ARRAY. It must not be scalar.
- **INDX1, ..., INDXn** must be of type integer and conformable with ARRAY. The number of INDX arguments must be equal to the rank of BASE.
- **MASK (optional)** must be of type logical and must be conformable with ARRAY.

**Result Type, Type Parameter, and Shape.** Same as BASE.

**Result Value.** The element of the result corresponding to the element b of BASE has the value MAXVAL( (/a_1, a_2, ..., a_m, b/) ), where (a_1, ..., a_m) are the elements of ARRAY associated with b as described in Section 5.4.4.

**Example.** MAXVAL_SCATTER((/1, 2, 3, 1/), (/4, -5, 7/), (/1, 1, 2, 2/)) is [ 4 3 7 ].

5.7.33  MAXVAL_SUFFIX(ARRAY, DIM, MASK, SEGMENT, EXCLUSIVE)

**Optional Arguments.** DIM, MASK, SEGMENT, EXCLUSIVE

**Description.** Computes a reverse, segmented MAXVAL scan along dimension DIM of ARRAY.

**Class.** Transformational function.

**Arguments.**

- **ARRAY** must be of type integer or real. It must not be scalar.
- **DIM (optional)** must be scalar and of type integer with a value in the range 1 ≤ DIM ≤ n, where n is the rank of ARRAY.
- **MASK (optional)** must be of type logical and must be conformable with ARRAY.
- **SEGMENT (optional)** must be of type logical and must have the same shape as ARRAY.
5.7. SPECIFICATIONS OF LIBRARY PROCEDURES

EXCLUSIVE (optional) must be of type logical and must be scalar.

Result Type, Type Parameter, and Shape. Same as ARRAY.

Result Value. Element $r$ of the result has the value $\text{MAXVAL}((/ a_1, \ldots, a_m /))$ where $(a_1, \ldots, a_m)$ is the (possibly empty) set of elements of ARRAY selected to contribute to $r$ by the rules stated in Section 5.4.5.

Example. $\text{MAXVAL\_SUFFIX}( (/3,4,-5,2,5/), \text{SEGMENT}=(/F,F,F,T,T/) )$ is $[4 4 -5 5 5]$.

5.7.34 MINVAL\_PREFIX(ARRAY, DIM, MASK, SEGMENT, EXCLUSIVE)

Optional Arguments. DIM, MASK, SEGMENT, EXCLUSIVE

Description. Computes a segmented MINVAL scan along dimension DIM of ARRAY.

Class. Transformational function.

Arguments.

ARRAY must be of type integer or real. It must not be scalar.

DIM (optional) must be scalar and of type integer with a value in the range $1 \leq \text{DIM} \leq n$, where $n$ is the rank of ARRAY.

MASK (optional) must be of type logical and must be conformable with ARRAY.

SEGMENT (optional) must be of type logical and must have the same shape as ARRAY.

EXCLUSIVE (optional) must be of type logical and must be scalar.

Result Type, Type Parameter, and Shape. Same as ARRAY.

Result Value. Element $r$ of the result has the value $\text{MINVAL}((/ a_1, \ldots, a_m /))$ where $(a_1, \ldots, a_m)$ is the (possibly empty) set of elements of ARRAY selected to contribute to $r$ by the rules stated in Section 5.4.5.

Example. $\text{MINVAL\_PREFIX}( (/1,2,-3,4,5/), \text{SEGMENT}=(/F,F,F,T,T/) )$ is $[1 1 -3 4 4]$.

5.7.35 MINVAL\_SCATTER(ARRAY,BASE,INDX1, ..., INDXn, MASK)

Optional Argument. MASK

Description. Scatters elements of ARRAY selected by MASK to positions of the result indicated by index arrays INDX1, ..., INDXn. Each element of the result is assigned the maximum value of the corresponding element of BASE and the elements of ARRAY scattered to that position.

Class. Transformational function.
Arguments.

ARRAY must be of type integer or real. It must not be scalar.
BASE must be of the same type and kind type parameter as ARRAY. It must not be scalar.
INDX1,...,INDXn must be of type integer and conformable with ARRAY. The number of INDX arguments must be equal to the rank of BASE.
MASK (optional) must be of type logical and must be conformable with ARRAY.

Result Type, Type Parameter, and Shape. Same as BASE.

Result Value. The element of the result corresponding to the element b of BASE has the value MINVAL(/a1,a2,...,am,b/), where (a1,...,am) are the elements of ARRAY associated with b as described in Section 5.4.4.

Example. MINVAL.SCATTER(/1,-2,-3,6/, /4,3,7/, /1,1,2,2/) is [-2 -3 7].

5.7.36 MINVAL.SUFFIX(ARRAY, DIM, MASK, SEGMENT, EXCLUSIVE)

Optional Arguments. DIM, MASK, SEGMENT, EXCLUSIVE

Description. Computes a reverse, segmented MINVAL scan along dimension DIM of ARRAY.

Class. Transformational function.

Arguments.

ARRAY must be of type integer or real. It must not be scalar.
DIM (optional) must be scalar and of type integer with a value in the range 1 \leq DIM \leq n, where n is the rank of ARRAY.
MASK (optional) must be of type logical and must be conformable with ARRAY.
SEGMENT (optional) must be of type logical and must have the same shape as ARRAY.
EXCLUSIVE (optional) must be of type logical and must be scalar.

Result Type, Type Parameter, and Shape. Same as ARRAY.

Result Value. Element r of the result has the value MINVAL(/a1,...,am/) where (a1,...,am) is the (possibly empty) set of elements of ARRAY selected to contribute to r by the rules stated in Section 5.4.5.

Example. MINVAL.SUFFIX( /1,2,-3,4,5/, SEGMENT= /F,F,F,T,T/ ) is [-3 -3 -3 4 5].
5.7.37 PARITY(MASK, DIM)

Optional Argument. DIM

Description. Determine whether an odd number of values are true in MASK along dimension DIM.

Class. Transformational function.

Arguments.

MASK must be of type logical. It must not be scalar.

DIM (optional) must be scalar and of type integer with a value in the range $1 \leq \text{DIM} \leq n$, where $n$ is the rank of MASK. The corresponding actual argument must not be an optional dummy argument.

Result Type, Type Parameter, and Shape. The result is of type logical with the same kind type parameter as MASK. It is scalar if DIM is absent or if MASK has rank one; otherwise, the result is an array of rank $n - 1$ and shape $(d_1, d_2, \ldots, d_{\text{DIM}-1}, d_{\text{DIM}+1}, \ldots, d_n)$ where $(d_1, d_2, \ldots, d_n)$ is the shape of MASK.

Result Value.

Case (i): The result of PARITY(MASK) is the .NEQV. reduction of all the elements of MASK. If MASK has size zero, the result has the value false. See Section 5.4.3.

Case (ii): If MASK has rank one, PARITY(MASK, DIM=1) has a value equal to that of PARITY(MASK). Otherwise, the value of element $(s_1, s_2, \ldots, s_{\text{DIM}-1}, s_{\text{DIM}+1}, \ldots, s_n)$ of PARITY(MASK, DIM=1) is equal to PARITY(MASK($s_1, s_2, \ldots, s_{\text{DIM}-1}, s_{\text{DIM}+1}, \ldots, s_n$)).

Examples.

Case (i): The value of PARITY($(/T, T, T, F/)$) is true.

Case (ii): If B is the array $\begin{bmatrix} T & T & F \\ T & T & T \end{bmatrix}$, then PARITY(B, DIM = 1) is $\begin{bmatrix} F & F & T \end{bmatrix}$ and PARITY(B, DIM = 2) is $\begin{bmatrix} F & T \end{bmatrix}$.

5.7.38 PARITY_PREFIX(MASK, DIM, SEGMENT, EXCLUSIVE)

Optional Arguments. DIM, SEGMENT, EXCLUSIVE

Description. Computes a segmented logical exclusive OR scan along dimension DIM of MASK.

Class. Transformational function.

Arguments.

MASK must be of type logical. It must not be scalar.
DIM (optional) must be scalar and of type integer with a value in the range $1 \leq \text{DIM} \leq n$, where $n$ is the rank of MASK.

SEGMENT (optional) must be of type logical and must have the same shape as MASK.

EXCLUSIVE (optional) must be of type logical and must be scalar.

Result Type, Type Parameter, and Shape. Same as MASK.

Result Value. Element $r$ of the result has the value $\text{PARITY}(\langle a_1, \ldots, a_m \rangle)$ where $(a_1, \ldots, a_m)$ is the (possibly empty) set of elements of MASK selected to contribute to $r$ by the rules stated in Section 5.4.5.

Example. $\text{PARITY}_\text{PREFIX}(\langle T,F,T,T,T \rangle, \text{SEGMENT}=\langle F,F,F,T,T \rangle)$ is $\begin{bmatrix} T & T & F & T & F \end{bmatrix}$.

5.7.39 \text{PARITY}_\text{SCATTER}(\text{MASK}, \text{BASE}, \text{INDX}_1, \ldots, \text{INDX}_n)$

Description. Scatters elements of MASK to positions of the result indicated by index arrays $\text{INDX}_1, \ldots, \text{INDX}_n$. An element of the result is true if and only if the number of true values among the corresponding element of BASE and the elements of MASK scattered to that position is odd.

Class. Transformational function.

Arguments.

\text{MASK} must be of type logical. It must not be scalar.

\text{BASE} must be of type logical with the same kind type parameter as MASK. It must not be scalar.

$\text{INDX}_1, \ldots, \text{INDX}_n$ must be of type integer and conformable with MASK. The number of INDX arguments must be equal to the rank of BASE.

Result Type, Type Parameter, and Shape. Same as BASE.

Result Value. The element of the result corresponding to the element $b$ of BASE has the value $\text{PARITY}(\langle a_1, a_2, \ldots, a_m, b \rangle)$, where $(a_1, \ldots, a_m)$ are the elements of MASK associated with $b$ as described in Section 5.4.4.

Example. $\text{PARITY}_\text{SCATTER}(\langle T,T,T,T \rangle, \langle T,F,F \rangle, \langle 1,1,1,2 \rangle)$ is $\begin{bmatrix} F & T & F \end{bmatrix}$.

5.7.40 \text{PARITY}_\text{SUFFIX}(\text{MASK}, \text{DIM}, \text{SEGMENT}, \text{EXCLUSIVE})

Optional Arguments. DIM, SEGMENT, EXCLUSIVE

Description. Computes a reverse, segmented logical exclusive OR scan along dimension DIM of MASK.
5.7. **SPECIFICATIONS OF LIBRARY PROCEDURES**

Class. Transformational function.

Arguments.

**MASK** must be of type logical. It must not be scalar.

**DIM (optional)** must be scalar and of type integer with a value in the range \(1 \leq \text{DIM} \leq n\), where \(n\) is the rank of **MASK**.

**SEGMENT (optional)** must be of type logical and must have the same shape as **MASK**.

**EXCLUSIVE (optional)** must be of type logical and must be scalar.

Result Type, Type Parameter, and Shape. Same as **MASK**.

Result Value. Element \(r\) of the result has the value \(\text{PARITY}(/ a_1, \ldots, a_m /)\) where \((a_1, \ldots, a_m)\) is the (possibly empty) set of elements of **MASK** selected to contribute to \(r\) by the rules stated in Section 5.4.5.

Example. \(\text{PARITY\_SUFFIX}(/T,F,T,T,T/), \text{SEGMENT}=(/F,F,F,T,T/)\) is \([F\ T\ T\ F\ T]\).

5.7.41 **POPCNT(I)**

Description. Return the number of one bits in an integer.

Class. Elemental function.

Argument. I must be of type integer.

Result Type and Type Parameter. Same as I.

Result Value. **POPCNT(I)** is the number of one bits in the binary representation of the integer I. The model for the interpretation of an integer as a sequence of bits is in Section 13.5.7 of the Fortran 90 Standard.

Example. **POPCNT(I)** = **COUNT**(/ (**BTEST**(I,J), J=0, **BIT\_SIZE**(I)-1) /)), for scalar I.

5.7.42 **POPPAR(I)**

Description. Return the parity of an integer.

Class. Elemental function.

Argument. I must be of type integer.

Result Type and Type Parameter. Same as I.

Result Value. **POPPAR(I)** is 1 if there are an odd number of one bits in I and zero if there are an even number. The model for the interpretation of an integer as a sequence of bits is in Section 13.5.7 of the Fortran 90 Standard.

Example. For scalar I, \(\text{POPPAR}(x) = \text{MERGE}(1,0,\text{BTEST}(\text{POPCNT}(x),0))\).
5.7.43 **PRODUCT PREFIX(ARRAY, DIM, MASK, SEGMENT, EXCLUSIVE)**

*Optional Arguments.* DIM, MASK, SEGMENT, EXCLUSIVE

*Description.* Computes a segmented PRODUCT scan along dimension DIM of ARRAY.

*Class.* Transformational function.

*Arguments.*

- **ARRAY** must be of type integer, real, or complex. It must not be scalar.

- **DIM (optional)** must be scalar and of type integer with a value in the range \(1 \leq \text{DIM} \leq n\), where \(n\) is the rank of ARRAY.

- **MASK (optional)** must be of type logical and must be conformable with ARRAY.

- **SEGMENT (optional)** must be of type logical and must have the same shape as ARRAY.

- **EXCLUSIVE (optional)** must be of type logical and must be scalar.

*Result Type, Type Parameter, and Shape.* Same as ARRAY.

*Result Value.* Element \(r\) of the result has the value \(\text{PRODUCT}(\langle / \ a_1, \ldots, a_m \ / \rangle)\) where \(\langle a_1, \ldots, a_m \rangle\) is the (possibly empty) set of elements of ARRAY selected to contribute to \(r\) by the rules stated in Section 5.4.5.

*Example.* \(\text{PRODUCT} \ (\langle / 1,2,3,4,5 / \rangle, \ \text{SEGMENT}= \langle / F,F,F,T,T / \rangle) \) is

\[
\begin{bmatrix}
1 & 2 & 6 & 4 & 20
\end{bmatrix}.
\]

5.7.44 **PRODUCT SCATTER(ARRAY, BASE, INDX1, ..., INDXn, MASK)**

*Optional Argument.* MASK

*Description.* Scatters elements of ARRAY selected by MASK to positions of the result indicated by index arrays INDX1, ..., INDXn. Each element of the result is equal to the product of the corresponding element of BASE and the elements of ARRAY scattered to that position.

*Class.* Transformational function.

*Arguments.*

- **ARRAY** must be of type integer, real, or complex. It must not be scalar.

- **BASE** must be of the same type and kind type parameter as ARRAY. It must not be scalar.

- **INDX1, ..., INDXn** must be of type integer and conformable with ARRAY. The number of INDX arguments must be equal to the rank of BASE.
5.7. **SPECIFICATIONS OF LIBRARY PROCEDURES**

MASK (optional) must be of type logical and must be conformable with ARRAY.

Result Type, Type Parameter, and Shape. Same as BASE.

Result Value. The element of the result corresponding to the element $b$ of BASE has the value $\text{PRODUCT}((a_1, a_2, ..., a_m, b))$, where $(a_1, ..., a_m)$ are the elements of ARRAY associated with $b$ as described in Section 5.4.4.

Example. $\text{PRODUCT\_SCATTER}((1, 2, 3, 1), (4, -5, 7), (1, 1, 2, 2))$ is $\begin{bmatrix} 8 & -15 & 7 \end{bmatrix}$.

5.7.45 **PRODUCT\_SUFFIX(ARRAY, DIM, MASK, SEGMENT, EXCLUSIVE)**

Optional Arguments. DIM, MASK, SEGMENT, EXCLUSIVE

Description. Computes a reverse, segmented PRODUCT scan along dimension DIM of ARRAY.

Class. Transformational function.

Arguments.

ARRAY must be of type integer, real, or complex. It must not be scalar.

DIM (optional) must be scalar and of type integer with a value in the range $1 \leq \text{DIM} \leq n$, where $n$ is the rank of ARRAY.

MASK (optional) must be of type logical and must be conformable with ARRAY.

SEGMENT (optional) must be of type logical and must have the same shape as ARRAY.

EXCLUSIVE (optional) must be of type logical and must be scalar.

Result Type, Type Parameter, and Shape. Same as ARRAY.

Result Value. Element $r$ of the result has the value $\text{PRODUCT}((a_1, ..., a_m))$ where $(a_1, ..., a_m)$ is the (possibly empty) set of elements of ARRAY selected to contribute to $r$ by the rules stated in Section 5.4.5.

Example. $\text{PRODUCT\_SUFFIX}((1, 2, 3, 4, 5), \text{SEGMENT} = (\text{F, F, F, T, T/})$ is $\begin{bmatrix} 6 & 6 & 3 & 20 & 5 \end{bmatrix}$.

5.7.46 **SUM\_PREFIX(ARRAY, DIM, MASK, SEGMENT, EXCLUSIVE)**

Optional Arguments. DIM, MASK, SEGMENT, EXCLUSIVE

Description. Computes a segmented SUM scan along dimension DIM of ARRAY.

Class. Transformational function.
Arguments.

ARRAY must be of type integer, real, or complex. It must not be scalar.

DIM (optional) must be scalar and of type integer with a value in the range \(1 \leq \text{DIM} \leq n\), where \(n\) is the rank of ARRAY.

MASK (optional) must be of type logical and must be conformable with ARRAY.

SEGMENT (optional) must be of type logical and must have the same shape as ARRAY.

EXCLUSIVE (optional) must be of type logical and must be scalar.

Result Type, Type Parameter, and Shape. Same as ARRAY.

Result Value. Element \(r\) of the result has the value \(\text{SUM}( \langle / a_1, \ldots, a_m \rangle )\) where \((a_1, \ldots, a_m)\) is the (possibly empty) set of elements of ARRAY selected to contribute to \(r\) by the rules stated in Section 5.4.5.

Example. \(\text{SUM}_{\text{PREFIX}}( \langle /1,2,3,4,5\rangle, \text{SEGMENT}= \langle /F,F,F,T,T\rangle )\) is \([1\ 3\ 6\ 4\ 9]\).

5.7.47 \text{SUM}_\text{SCATTER}(\text{ARRAY},\text{BASE},\text{INDX}_1, \ldots, \text{INDX}_n, \text{MASK})

Optional Argument. MASK

Description. Scatters elements of ARRAY selected by MASK to positions of the result indicated by index arrays INDX\(_1\), \ldots, INDX\(_n\). Each element of the result is equal to the sum of the corresponding element of BASE and the elements of ARRAY scattered to that position.

Class. Transformational function.

Arguments.

ARRAY must be of type integer, real, or complex. It must not be scalar.

BASE must be of the same type and kind type parameter as ARRAY. It must not be scalar.

INDX\(_1\), \ldots, INDX\(_n\) must be of type integer and conformable with ARRAY. The number of INDX arguments must be equal to the rank of BASE.

MASK (optional) must be of type logical and must be conformable with ARRAY.

Result Type, Type Parameter, and Shape. Same as BASE.

Result Value. The element of the result corresponding to the element \(b\) of BASE has the value \(\text{SUM}( \langle / a_1, a_2, \ldots, a_m, b \rangle )\), where \((a_1, \ldots, a_m)\) are the elements of ARRAY associated with \(b\) as described in Section 5.4.4.

Example. \(\text{SUM}_\text{SCATTER}(\langle /1, 2, 3, 1\rangle, \langle /4, -5, 7\rangle, \langle /1, 1, 2, 2\rangle)\) is \([7 -1 7]\).
5.7. SPECIFICATIONS OF LIBRARY PROCEDURES

5.7.48 SUM.SUFFIX(ARRAY, DIM, MASK, SEGMENT, EXCLUSIVE)

Optional Arguments. DIM, MASK, SEGMENT, EXCLUSIVE

Description. Computes a reverse, segmented SUM scan along dimension DIM of ARRAY.

Class. Transformational function.

Arguments.

ARRAY must be of type integer, real, or complex. It must not be scalar.

DIM (optional) must be scalar and of type integer with a value in the range $1 \leq \text{DIM} \leq n$, where $n$ is the rank of ARRAY.

MASK (optional) must be of type logical and must be conformable with ARRAY.

SEGMENT (optional) must be of type logical and must have the same shape as ARRAY.

EXCLUSIVE (optional) must be of type logical and must be scalar.

Result Type, Type Parameter, and Shape. Same as ARRAY.

Result Value. Element $r$ of the result has the value $\text{SUM}((/ a_1, \ldots, a_m /))$ where $(a_1, \ldots, a_m)$ is the (possibly empty) set of elements of ARRAY selected to contribute to $r$ by the rules stated in Section 5.4.5.

Example. SUM.SUFFIX( /(1,2,3,4,5/) , SEGMENT= /(F,F,F,T,T/) ) is

\[
\begin{bmatrix}
6 & 5 & 3 & 9 & 5
\end{bmatrix}.
\]