TS 18508 Additional Parallel Features in Fortran

WG5/N2007

12th March 2014 9:46

Draft document for WG5 Ballot

(Blank page)

Contents

1 Scope . . . 1

2 Normative references . . . 3

3 Terms and definitions . . . 5

4 Compatibility . . . 7
- 4.1 New intrinsic procedures . . . 7
- 4.2 Fortran 2008 compatibility . . . 7

5 Teams of images . . . 9
- 5.1 Introduction . . . 9
- 5.2 TEAM_TYPE . . . 9
- 5.3 CHANGE TEAM construct . . . 9
- 5.4 Image selectors . . . 11
- 5.5 FORM TEAM statement . . . 11
- 5.6 SYNC TEAM statement . . . 12
- 5.7 FAIL IMAGE statement . . . 13
- 5.8 STAT_FAILED_IMAGE . . . 13

6 Events . . . 15
- 6.1 Introduction . . . 15
- 6.2 EVENT_TYPE . . . 15
- 6.3 EVENT POST statement . . . 15
- 6.4 EVENT WAIT statement . . . 16

7 Intrinsic procedures . . . 17
- 7.1 General . . . 17
- 7.2 Atomic subroutines . . . 17
- 7.3 Collective subroutines . . . 17
- 7.4 New intrinsic procedures . . . 18
  - 7.4.1 ATOMIC_ADD (ATOM, VALUE) or ATOMIC_ADD (ATOM, VALUE, OLD) . . . 18
  - 7.4.2 ATOMIC_AND (ATOM, VALUE) or ATOMIC_AND (ATOM, VALUE, OLD) . . . 18
  - 7.4.3 ATOMIC_CAS (ATOM, OLD, COMPARE, NEW) . . . 19
  - 7.4.4 ATOMIC_OR (ATOM, VALUE) or ATOMIC_OR (ATOM, VALUE, OLD) . . . 19
  - 7.4.5 ATOMIC_XOR (ATOM, VALUE) or ATOMIC_XOR (ATOM, VALUE, OLD) . . . 20
  - 7.4.6 CO_BROADCAST (SOURCE, SOURCE_IMAGE [, STAT, ERRMSG]) . . . 20
  - 7.4.7 CO_MAX (SOURCE [, RESULT, RESULT_IMAGE, STAT, ERRMSG]) . . . 20
  - 7.4.8 CO_MIN (SOURCE [, RESULT, RESULT_IMAGE, STAT, ERRMSG]) . . . 21
  - 7.4.9 CO_REDUCE (SOURCE, OPERATOR [, RESULT, RESULT_IMAGE, STAT, ERRMSG]) 22
  - 7.4.10 CO_SUM (SOURCE [, RESULT, RESULT_IMAGE, STAT, ERRMSG]) . . . 22
  - 7.4.11 EVENT_QUERY (EVENT, COUNT [, STAT, ERRMSG]) . . . 23
  - 7.4.12 FAILED_IMAGES ([TEAM, KIND]) . . . 24
  - 7.4.13 GET_TEAM (TEAM_VAR [,DISTANCE]) . . . 24
  - 7.4.14 TEAM_DEPTH( ) . . . 25
  - 7.4.15 TEAM_ID ([DISTANCE]) . . . 25
- 7.5 Modified intrinsic procedures . . . 26

7.5.1 MOVE_ALLOC . . . 26
7.5.2 NUM_IMAGES . . . 27
7.5.3 THIS_IMAGE . . . 27

8 Required editorial changes to ISO/IEC 1539-1:2010(E) . . . 29
8.1 General . . . 29
8.2 Edits to Introduction . . . 29
8.3 Edits to clause 1 . . . 29
8.4 Edits to clause 2 . . . 30
8.5 Edits to clause 4 . . . 31
8.6 Edits to clause 6 . . . 31
8.7 Edits to clause 8 . . . 32
8.8 Edits to clause 13 . . . 33
8.9 Edits to clause 16 . . . 36
8.10 Edits to annex A . . . 36
8.11 Edits to annex C . . . 37

Annex A (informative) Extended notes . . . 39
A.1 Clause 5 notes . . . 39
A.1.1 Example using three teams . . . 39
A.1.2 Example involving failed images . . . 39
A.1.3 Accessing coarrays in sibling teams . . . 41
A.1.4 Reducing the codimension of a coarray . . . 42
A.2 Clause 6 notes . . . 43
A.2.1 EVENT_QUERY example . . . 43
A.2.2 EVENTS example . . . 44
A.3 Clause 7 notes . . . 45
A.3.1 Collective subroutine examples . . . 45
A.3.2 Atomic memory consistency . . . 45

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and nongovernmental, in liaison with ISO and IEC, also take part in the work. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

The main task of the joint technical committee is to prepare International Standards. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

In other circumstances, particularly when there is an urgent market requirement for such documents, the joint technical committee may decide to publish an ISO/IEC Technical Specification (ISO/IEC TS), which represents an agreement between the members of the joint technical committee and is accepted for publication if it is approved by 2/3 of the members of the committee casting a vote.

An ISO/IEC TS is reviewed after three years in order to decide whether it will be confirmed for a further three years, revised to become an International Standard, or withdrawn. If the ISO/IEC TS is confirmed, it is reviewed again after a further three years, at which time it must either be transformed into an International Standard or be withdrawn.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

ISO/IEC TS 18508:2014 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC22, *Programming languages, their environments and system software interfaces.*

Introduction

The system for parallel programming in Fortran, as standardized by ISO/IEC 1539-1:2010, defines simple syntax for access to data on another image of a program, a set of synchronization statements for controlling the ordering of execution segments between images, and collective allocation and deallocation of memory on all images.

The existing system for parallel programming does not provide for an environment where a subset of the images can easily work on part of an application while not affecting other images in the program. This complicates development of independent parts of an application by separate teams of programmers. The existing system does not provide a mechanism for a processor to identify what images have failed during execution of a program. This adversely affects the resilience of programs executing on large systems. The synchronization primitives available in the existing system do not provide for a convenient mechanism for ordering execution segments on different images without requiring that those images arrive at a synchronization point before either is allowed to progress. This introduces unnecessary inefficiency into programs. Finally, the existing system does not provide intrinsic procedures for commonly used collective and atomic memory operations. Intrinsic procedures for these operations can be highly optimized for the target computational system, providing significantly improved program performance.

This Technical Specification extends the facilites of Fortran for parallel programming to provide for grouping the images of a program into nonoverlapping teams that can more effectively execute independently parts of a larger problem, for the processor to indicate which images have failed during execution and allow continued execution of the program on the remaining images, for a system of events that can be used for fine grain ordering of execution segments, and for sets of collective and atomic memory operation subroutines that can provide better performance for specific operations involving more than one image.

The facility specified in this Technical Specification is a compatible extension of Fortran as standardized by ISO/IEC 1539-1:2010 and ISO/IEC 1539-1:2010/Cor 2:2013.

It is the intention of ISO/IEC JTC 1/SC22 that the semantics and syntax specified by this Technical Specification be included in the next revision of ISO/IEC 1539-1 without change unless experience in the implementation and use of this feature identifies errors that need to be corrected, or changes are needed to achieve proper integration, in which case every reasonable effort will be made to minimize the impact of such changes on existing implementations.

This Technical Specification is organized in 8 clauses:

Scope	Clause 1
Normative references	Clause 2
Terms and definitions	Clause 3
Compatibility	Clause 4
Teams of images	Clause 5
Events	Clause 6
Intrinsic procedures	Clause 7
Required editorial changes to ISO/IEC 1539-1:2010(E)	Clause 8

It also contains the following nonnormative material:

Extended notes	Annex A

1 Scope

This Technical Specification specifies the form and establishes the interpretation of facilities that extend the Fortran language defined by ISO/IEC 1539-1:2010 and ISO/IEC 1539-1:2010/Cor 2:2013. The purpose of this Technical Specification is to promote portability, reliability, maintainability, and efficient execution of parallel programs written in Fortran, for use on a variety of computing systems.

(Blank page)

2 Normative references

The following referenced standards are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO/IEC 1539-1:2010, *Information technology—Programming languages—Fortran—Part 1:Base language*

ISO/IEC 1539-1:2010/Cor 2:2013, *Information technology—Programming languages—Fortran—Part 1:Base language TECHNICAL CORRIGENDUM 2*

(Blank page)

3 Terms and definitions

For the purposes of this document, the terms and definitions given in ISO/IEC 1539-1:2010 and the following apply. The intrinsic module ISO_FORTRAN_ENV is extended by this Technical Specification.

3.1
collective subroutine
intrinsic subroutine that is invoked on the current team of images to perform a calculation on those images and assign the computed value on one or all of them (7.3)

3.2
team
set of images that can readily execute independently of other images (5.1)

3.2.1
current team
the team specified in the CHANGE TEAM statement of the innermost executing CHANGE TEAM construct, or the initial team if no CHANGE TEAM construct is active (5.1)

3.2.2
initial team
the current team when the program began execution (5.1)

3.2.3
parent team
team from which the current team was formed by executing a FORM TEAM statement (5.1)

3.2.4
team identifier
integer value identifying a team (5.1)

3.2.5
team distance
the distance between a team and one of its ancestors (5.1)

3.3
failed image
an image for which references or definitions of a variable on the image fail when that variable should be accessible, or the image fails to respond during the execution of an image control statement or a reference to a collective subroutine (5.8)

3.4
event variable
scalar variable of type EVENT_TYPE (6.2) in the intrinsic module ISO_FORTRAN_ENV

3.5
team variable
scalar variable of type TEAM_TYPE (5.2) in the intrinsic module ISO_FORTRAN_ENV

(Blank page)

4 Compatibility

4.1 New intrinsic procedures

This Technical Specification defines intrinsic procedures in addition to those specified in ISO/IEC 1539-1:2010. Therefore, a Fortran program conforming to ISO/IEC 1539-1:2010 might have a different interpretation under this Technical Specification if it invokes an external procedure having the same name as one of the new intrinsic procedures, unless that procedure is specified to have the EXTERNAL attribute.

4.2 Fortran 2008 compatibility

This Technical Specification specifies an upwardly compatible extension to ISO/IEC 1539-1:2010.

(Blank page)

5 Teams of images

5.1 Introduction

A team of images is a set of images that can readily execute independently of other images. Syntax and semantics of *image-selector* (R624 in ISO/IEC 1539-1:2010) have been extended to determine how cosubscripts are mapped to image indices for both sibling and ancestor team references. Initially, the current team consists of all the images and this is known as the initial team. A team is divided into new teams by executing a FORM TEAM statement. Each new team is identified by an integer value known as its team identifier. Information about the team to which the current image belongs can be determined by the processor from the collective value of the team variables on the images of the team.

Team distance is a measure of the distance between two teams, one of which is an ancestor of the other. The team distance between a team and itself is zero. Except for the initial team, every team has a unique parent team. The team distance between a team and its parent is one. The team distance between a team T and the parent of team A, which is an ancestor of T, is one more than the team distance between teams T and A.

The current team is the team specified in the CHANGE TEAM statement of the innermost executing CHANGE TEAM construct, or the initial team if no CHANGE TEAM construct is active.

A nonallocatable coarray that is neither a dummy argument, host associated with a dummy argument, declared as a local variable of a subprogram, nor declared in a BLOCK construct is established in the initial team. An allocated allocatable coarray is established in the team in which it was allocated. An unallocated allocatable coarray is not established. An associating coarray is established in the team of its CHANGE TEAM block. A nonallocatable coarray that is a dummy argument or host associated with a dummy argument is established in the team in which the procedure was invoked. A nonallocatable coarray that is a local variable of a subprogram or host associated with a local variable of a subprogram is established in the team in which the procedure was invoked. A nonallocatable coarray declared in a BLOCK construct is established in the team in which the BLOCK statement was executed.

5.2 TEAM_TYPE

TEAM_TYPE is a derived type with private components. It is an extensible type with no type parameters. Each component is fully default initialized. A scalar variable of this type describes a team. TEAM_TYPE is defined in the intrinsic module ISO_FORTRAN_ENV.

A scalar variable of type TEAM_TYPE is a team variable. The default initial value of a team variable shall not represent any valid team.

5.3 CHANGE TEAM construct

The CHANGE TEAM construct changes the current team to which the executing image belongs.

```
R501  change-team-construct   is  change-team-stmt
                                      block
                                      end-change-team-stmt

R502  change-team-stmt        is  [ team-construct-name: ] CHANGE TEAM ( team-variable ■
                                  ■ [, coarray-association-list] [, sync-stat-list ] )

R503  coarray-association     is  codimension-decl => coselector-name
```

R504	*coselector*	**is**	*coarray*
R505	*end-change-team-stmt*	**is**	END TEAM [(*sync-stat-list*)] [*team-construct-name*]
R506	*team-variable*	**is**	*scalar-variable*

C501 (R501) A branch within a CHANGE TEAM construct shall not have a branch target that is outside the construct.

C502 (R501) A RETURN statement shall not appear within a CHANGE TEAM construct.

C503 (R501) A *exit-stmt* or *cycle-stmt* within a CHANGE TEAM construct shall not belong to an outer construct.

C504 (R501) If the *change-team-stmt* of a *change-team-construct* specifies a *team-construct-name*, the corresponding *end-change-team-stmt* shall specify the same *team-construct-name*. If the *change-team-stmt* of a *change-team-construct* does not specify a *team-construct-name*, the corresponding *end-change-team-stmt* shall not specify a *team-construct-name*.

C505 (R503) The *coarray-name* in the *codimension-decl* shall not be the same as any *coselector-name* in the *change-team-stmt*.

C506 (R506) A *team-variable* shall be of the type TEAM_TYPE (5.2).

C507 (R502) No *coselector-name* shall appear more than once in a *change-team-stmt*.

A *coselector* shall be established when the CHANGE TEAM statement begins execution.

The *team-variable* shall have been defined by execution of a FORM TEAM statement in the team that executes the CHANGE TEAM statement or be the value of a team variable for the initial team. The values of the *team-variables* on the images of the team shall be those defined by execution of the same FORM TEAM statement on all the images of the team. The current team for the statements of the CHANGE TEAM *block* is the team specified by the value of the *team-variable*. The current team is not changed by a redefinition of the team variable during execution of the CHANGE TEAM construct.

A *codimension-decl* in a *coarray-association* associates a coarray with an established coarray during the execution of the block. This coarray is an associating entity (8.1.3.2, 8.1.3.3, 16.5.1.6 of ISO/IEC 1539-1:2010). Its name is an associate name that has the scope of the construct. It has the declared type, dynamic type, type parameters, rank, and bounds of the coselector. Apart from the final upper cobound, its corank and cobounds are those specified in the *codimension-decl*.

Within a CHANGE TEAM construct, a coarray that does not appear in a *coarray-association* has the corank and cobounds that it had when it was established.

An allocatable coarray that was allocated when execution of a CHANGE TEAM construct began shall not be deallocated during the execution of the construct. An allocatable coarray that is allocated when execution of a CHANGE TEAM construct completes is deallocated if it was not allocated when execution of the construct began.

The CHANGE TEAM and END TEAM statements are image control statements. All nonfailed images of the team containing the executing image that is identified by *team-variable* shall execute the same CHANGE TEAM statement. When a CHANGE TEAM statement is executed, there is an implicit synchronization of all nonfailed images of the team containing the executing image that is identified by *team-variable*. On each nonfailed image of the team, execution of the segment following the statement is delayed until all the other nonfailed images of the team have executed the same statement the same number of times. When a CHANGE TEAM construct completes execution, there is an implicit synchronization of all nonfailed images in the current team. On each nonfailed image of the team, execution of the segment following the END TEAM statement is delayed until all the other nonfailed images of the team have executed the same construct the same number of times.

NOTE 5.1

Deallocation of an allocatable coarray that was not allocated at the beginning of a CHANGE TEAM construct, but is allocated at the end of execution of the construct, occurs even for allocatable coarrays with the SAVE attribute.

5.4 Image selectors

The syntax rule R624 *image-selector* in subclause 6.6 of ISO/IEC 1539-1:2010 is replaced by:

R624 *image-selector* **is** *lbracket* [*team-variable* ::] *cosubscript-list* ■
■ [, TEAM_ID = *scalar-int-expr*] *rbracket*

C508 (R624) *team-variable* and TEAM_ID = shall not both appear in the same *image-selector*.

If *team-variable* appears in a coarray designator, it shall be defined with a value that represents an ancestor of the current team. The coarray shall be established in that team or an ancestor of that team and the cosubscripts determine an image index in that team.

If TEAM_ID = appears in a coarray designator, the *scalar-int-expr* shall be defined with the value of a team identifier for one of the teams that were formed by the execution of the FORM TEAM statement for the current team. The coarray shall be established in an ancestor of the current team and the cosubscripts determine an image index in the team identified by TEAM_ID.

NOTE 5.2

In the following code, the vector *a* of length N*P is distributed over P images. Each has an array A(0:N+1) holding its own values of *a* and halo values from its two neighbors. The images are divided into two teams that execute independently but periodically exchange halo data. Before the data exchange, all the images (of the initial team) must be synchronized and for the data exchange the coindices of the initial team are needed.

```
USE, INTRINSIC :: ISO_FORTRAN_ENV
TYPE(TEAM_TYPE) :: INITIAL, BLOCK
REAL :: A(0:N+1)[*]
INTEGER :: ME, P2
CALL GET_TEAM(INITIAL)
ME = THIS_IMAGE()
P2 = NUM_IMAGES()/2
FORM TEAM(1+(ME-1)/P2,BLOCK)
CHANGE TEAM(BLOCK,B[*]=>A)
   DO
      ! Iterate within team
        :
      ! Halo exchange across team boundary
      SYNC TEAM(INITIAL)
      IF(ME==P2  ) B(N+1) = A(1)[INITIAL::ME+1]
      IF(ME==P2+1)   B(0) = A(N)[INITIAL::ME-1]
      SYNC TEAM(INITIAL)
   END DO
END TEAM
```

5.5 FORM TEAM statement

R507 *form-team-stmt* **is** FORM TEAM (*team-id*, *team-variable* ■
■ [, *form-team-spec-list*])

R508	*team-id*	**is**	*scalar-int-expr*
R509	*form-team-spec*	**is**	NEW_INDEX = *scalar-int-expr*
		or	*sync-stat*

C509 (R507) No specifier shall appear more than once in a *form-team-spec-list*.

The FORM TEAM statement defines *team-variable* for a new team. The value of *team-id* specifies the new team to which the executing image will belong. The value of *team-id* shall be greater than zero and is the same for all images that are members of the same team.

The value of the *scalar-int-expr* in a NEW_INDEX= specifier specifies the image index that the executing image will have in the team specified by *team-id*. It shall be greater than zero and less than or equal to the number of images in the team. Each image with the same value for *team-id* shall have a different value for the NEW_INDEX= specifier. If the NEW_INDEX= specifier does not appear, the image index that the executing image will have in the team specified by *team-id* is assigned by the processor.

The FORM TEAM statement is an image control statement. If the FORM TEAM statement is executed on one image, it shall be executed by the same statement on all nonfailed images of the current team. When a FORM TEAM statement is executed, there is an implicit synchronization of all nonfailed images in the current team. On these images, execution of the segment following the statement is delayed until all other nonfailed images in the current team have executed the same statement the same number of times. If an error condition other than detection of a failed image occurs, the team variable becomes undefined.

NOTE 5.3

Executing the statement

```
      FORM TEAM ( 2-MOD(ME,2), ODD_EVEN )
```

with `ME` an integer with value THIS_IMAGE() and `ODD_EVEN` of type TEAM_TYPE, divides the current team into two teams according to whether the image index is even or odd.

NOTE 5.4

When executing on P^2 images with corresponding coarrays on each image representing parts of a larger array spread over a P by P square, the following code establishes teams for the rows with image indices equal to the column indices.

```
USE, INTRINSIC :: ISO_FORTRAN_ENV
TYPE(TEAM_TYPE) :: ROW
REAL :: A[P,*]
INTEGER :: ME(2)
ME(:) = THIS_IMAGE(A)
FORM TEAM(ME(1),ROW,NEW_INDEX=ME(2))
```

5.6 SYNC TEAM statement

R510	*sync-team-stmt*	**is**	SYNC TEAM (*team-variable* [, *sync-stat-list*])

The SYNC TEAM statement is an image control statement. The value of *team-variable* shall have been established by execution of a FORM TEAM statement by the current team or an ancestor of the current team, or be the value of a team variable for the initial team. The values of the *team-variables* on the images of the team shall be those defined by execution of the same FORM TEAM statement on all the images of the team or shall be the values of the team variables for the initial team. Execution of a SYNC TEAM statement performs a synchronization of the executing image with each of the other nonfailed images of the team specified by *team-variable*. Execution on an image, M, of the segment following the SYNC TEAM statement is delayed until each nonfailed other image of the specified team has executed a SYNC TEAM statement specifying the same team as many times as has image M. The segments that executed before the SYNC TEAM statement on an image precede the segments that execute after the SYNC TEAM statement on another image.

NOTE 5.5

A SYNC TEAM statement performs a synchronization of images of a particular team whereas a SYNC ALL statement performs a synchronization of all images of the current team.

5.7 FAIL IMAGE statement

R511 *fail-image-stmt* **is** FAIL IMAGE [*stop-code*]

Execution of a FAIL IMAGE statement causes the executing image to behave as if it has failed. No further statements are executed by that image.

When an image executes a FAIL IMAGE statement, its stop code, if any, is made available in a processor-dependent manner.

NOTE 5.6

The FAIL IMAGE statement allows a program to test a recovery algorithm without experiencing an actual failure.

On a processor that does not have the ability to detect that an image has failed, execution of a FAIL IMAGE statement might provide a simulated failure environment that provides debug information.

In a piece of code that executes about once a second, invoking this subroutine on an image

```
SUBROUTINE FAIL
   REAL :: X
   CALL RANDOM_NUMBER(X)
   IF (X<0.001) FAIL IMAGE "Subroutine FAIL called"
END SUBROUTINE FAIL
```

will randomly cause that image to have an independent 1/1000 chance of failure every second.

5.8 STAT_FAILED_IMAGE

If the processor has the ability to detect that an image has failed, the value of the default integer scalar constant STAT_FAILED_IMAGE is positive and different from the value of STAT_STOPPED_IMAGE, STAT_LOCKED, STAT_LOCKED_OTHER_IMAGE, or STAT_UNLOCKED; otherwise, the value of STAT_FAILED_IMAGE is negative. If the processor has the ability to detect that an image of the current team has failed and does so, the value of STAT_FAILED_IMAGE is assigned to the variable specified in a STAT=specifier in an execution of an image control statement, or the STAT argument in an invocation of a collective procedure. A failed image is one for which references or definitions of a variable on the image fail when that variable should be accessible, or the image fails to respond during the execution of an image control statement or a reference to a collective subroutine. A failed image remains failed for the remainder of the program execution. If more than one nonzero status value is valid for the execution of a statement, the status variable is defined with a value other than STAT_FAILED_IMAGE. The conditions that cause an image to fail are processor dependent. STAT_FAILED_IMAGE is defined in the intrinsic module ISO_FORTRAN_ENV.

NOTE 5.7

A failed image is usually associated with a hardware failure of the processor, memory system, or interconnection network. A failure that occurs while a coindexed reference or definition, or collective action, is in progress may leave variables on other images that would be defined by that action in an undefined state. Similarly, failure while using a file may leave that file in an undefined state. A failure on one image may cause other images to fail for that reason.

NOTE 5.8

Continued execution after the failure of image 1 might be difficult because of the lost connection to standard input. However, the likelihood of a given image failing is small. With a large number of images, the likelihood of some image other than image 1 failing is significant and it is for this circumstance that STAT_FAILED_IMAGE is designed.

6 Events

6.1 Introduction

An image can post an event to notify another image that it can proceed to work on tasks that use common resources. An image can wait on events posted by other images and can query if images have posted events.

6.2 EVENT_TYPE

EVENT_TYPE is a derived type with private components. It is an extensible type with no type parameters. Each component is fully default initialized. EVENT_TYPE is defined in the intrinsic module ISO_FORTRAN_ENV .

A scalar variable of type EVENT_TYPE is an event variable. An event variable has a count that is updated by execution of a sequence of EVENT POST or EVENT WAIT statements. The effect of each change is as if it occurred instantaneously, without any overlap with another change. A coarray that is of type EVENT_TYPE may be referenced or defined during the execution of a segment that is unordered relative to the execution of another segment in which that coarray of type EVENT_TYPE is defined. The initial value of the event count of an event variable is zero. The processor shall support a maximum value of the event count of at least HUGE(0).

C601 A named variable of type EVENT_TYPE shall be a coarray. A named variable with a noncoarray subcomponent of type EVENT_TYPE shall be a coarray.

C602 An event variable shall not appear in a variable definition context except as the *event-variable* in an EVENT POST or EVENT WAIT statement, as an *allocate-object* in an ALLOCATE statement without a SOURCE= *alloc-opt*, as an *allocate-object* in a DEALLOCATE statement, or as an actual argument in a reference to a procedure with an explicit interface where the corresponding dummy argument has INTENT (INOUT).

C603 A variable with a subobject of type EVENT_TYPE shall not appear in a variable definition context except as an *allocate-object* in an ALLOCATE statement without a SOURCE= *alloc-opt*, as an *allocate-object* in a DEALLOCATE statement, or as an actual argument in a reference to a procedure with an explicit interface where the corresponding dummy argument has INTENT (INOUT).

6.3 EVENT POST statement

The EVENT POST statement provides a way to post an event. It is an image control statement.

R601	*event-post-stmt*	**is**	EVENT POST(*event-variable* [, *sync-stat-list*])
R602	*event-variable*	**is**	*scalar-variable*

C604 (R602) An *event-variable* shall be of the type EVENT_TYPE (6.2).

Successful execution of an EVENT POST statement increments the count of the event variable by 1. If an error condition occurs during the execution of an EVENT POST statement, the count does not change.

If the segment that precedes an EVENT POST statement is unordered with respect to the segment that precedes another EVENT POST statement for the same event variable, the order of execution of the EVENT POST statements is processor dependent.

NOTE 6.1

> It is expected that an image will continue executing after posting an event without waiting for an EVENT WAIT statement to execute on the image of the event variable.

6.4 EVENT WAIT statement

The EVENT WAIT statement provides a way to wait until an event is posted. It is an image control statement.

R603	*event-wait-stmt*	**is**	EVENT WAIT(*event-variable* [, *wait-spec-list*])
R604	*wait-spec*	**is**	UNTIL_COUNT = *scalar-int-expr*
		or	*sync-stat*

C605 (R603) An *event-variable* in an *event-wait-stmt* shall not be coindexed.

Execution of an EVENT WAIT statement causes the following sequence of actions:

(1) the threshold of its event argument is set to UNTIL_COUNT if this specifier is provided with a positive value, and to 1 otherwise,

(2) the executing image waits until the count of the event variable is greater than or equal to its threshold value or an error condition occurs, and

(3) if no error condition occurs, the event count is decreased by its threshold value.

If the count of an event variable increases because of the execution of an EVENT POST statement on image M and later in the sequence decreases because of the execution of an EVENT WAIT statement on image T, the segments preceding the EVENT POST statement on image M precede the segments following the EVENT WAIT statement on image T.

NOTE 6.2

> The segment that follows the execution of an EVENT WAIT statement is ordered with respect to all the segments that precede EVENT POST statements that caused prior changes in the sequence of values of the event variable.

NOTE 6.3

> Event variables of type EVENT_TYPE are restricted so that EVENT WAIT statements can only wait on an event variable on the executing image. This enables more efficient implementation of this concept.

7 Intrinsic procedures

7.1 General

Detailed specifications of the generic intrinsic procedures ATOMIC_ADD, ATOMIC_AND, ATOMIC_CAS, ATOMIC_OR, ATOMIC_XOR, CO_BROADCAST, CO_MAX, CO_MIN, CO_REDUCE, CO_SUM, EVENT_QUERY, FAILED_IMAGES, GET_TEAM, TEAM_DEPTH, and TEAM_ID are provided in 7.4. The types and type parameters of the arguments to these intrinsic procedures are determined by these specifications. The "Argument" paragraphs specify requirements on the actual arguments of the procedures. All of these intrinsic procedures are pure.

The intrinsic procedures MOVE_ALLOC, NUM_IMAGES, and THIS_IMAGE described in clause 13 of ISO/IEC 1539-1:2010 are extended as described in 7.5.

7.2 Atomic subroutines

An atomic subroutine is an intrinsic subroutine that performs an action on its ATOM argument atomically. The effect of executing atomic subroutines in unordered segments on a single atomic object is as if the subroutines were executed in some processor-dependent serial order, with none of the accesses to that object in any one subroutine execution interleaving with those in any other. The sequence of atomic actions within ordered segments is specified in 2.3.5 of ISO/IEC 1539-1:2010. For invocation of an atomic subroutine with an argument OLD, the determination of the value to be assigned to OLD is part of the atomic operation even though the assignment of that value to OLD is not. For invocation of an atomic subroutine, evaluation of an INTENT(IN) argument is not part of the atomic action.

This Technical Specification does not specify a formal data consistency model for atomic references. Developing a formal data consistency model is left until the integration of these facilities into ISO/IEC 1539-1.

7.3 Collective subroutines

A collective subroutine is one that is invoked on each nonfailed image of the current team to perform a calculation on those images and that assigns the computed value on one or all of them. If it is invoked by one image, it shall be invoked by the same statement on all nonfailed images of the current team in execution segments that are not ordered with respect to each other. From the beginning to the end of execution as the current team, the sequence of invocations of collective subroutines shall be the same on all nonfailed images of the current team. A call to a collective subroutine shall appear only in a context that allows an image control statement.

If the SOURCE or RESULT argument to a collective subroutine is a whole coarray the corresponding ultimate arguments on all images of the current team shall be corresponding coarrays as described in 2.4.7 of ISO/IEC 1539-1:2010.

Collective subroutines have the optional arguments STAT and ERRMSG. If the STAT argument is present in the invocation on one image it shall be present on the corresponding invocations on all of the images of the current team.

If the STAT argument is present in an invocation of a collective subroutine and its execution is successful, the argument is assigned the value zero.

If the STAT argument is present in an invocation of a collective subroutine and an error condition occurs, the argument is assigned a nonzero value, the RESULT argument becomes undefined if it is present, or the SOURCE argument becomes undefined otherwise. If execution involves synchronization with an image that has stopped, the argument is assigned the value of STAT_STOPPED_IMAGE in the intrinsic module ISO_FORTRAN_ENV; otherwise, if no image of the current team has stopped or failed, the argument is assigned a processor-dependent positive value that is different from the value of STAT_STOPPED_IMAGE or STAT_FAILED_IMAGE in the intrinsic module ISO_FORTRAN_ENV. If an image of the current team has been detected as failed, but no other error condition occurred, the argument is assigned the value of the constant STAT_FAILED_IMAGE.

If a condition occurs that would assign a nonzero value to a STAT argument but the STAT argument is not present, error termination is initiated.

If an ERRMSG argument is present in an invocation of a collective subroutine and an error condition occurs during its execution, the processor shall assign an explanatory message to the argument. If no such condition occurs, the processor shall not change the value of the argument.

NOTE 7.1

SOURCE becomes undefined in the event of an error condition for a collective with RESULT not present because it is intended that implementations be able to use SOURCE as scratch space.

NOTE 7.2

There is no separate synchronization at the beginning and end of an invocation of a collective procedure, which allows overlap with other actions. However, each collective involves transfer of data between images. The rules of Fortran do not allow the value of an associated argument such as SOURCE to be changed except via the argument. This includes action taken by another image that has not started its execution of the collective or has finished it. This restriction has the effect of a partial synchronization of invocations of a collective.

7.4 New intrinsic procedures

7.4.1 ATOMIC_ADD (ATOM, VALUE) or ATOMIC_ADD (ATOM, VALUE, OLD)

Description. Atomic add operation.

Class. Atomic subroutine.

Arguments.

ATOM	shall be a scalar coarray or coindexed object and of type integer with kind ATOMIC_INT_KIND, where ATOMIC_INT_KIND is a named constant in the intrinsic module ISO_FORTRAN_ENV. It is an INTENT (INOUT) argument. ATOM becomes defined with the value of ATOM + INT(VALUE, ATOMIC_INT_KIND).
VALUE	shall be scalar and of type integer. It is an INTENT (IN) argument.
OLD	shall be a scalar and of the same type and kind as ATOM. It is an INTENT (OUT) argument. It is defined with the value of ATOM that was used for performing the ADD operation.

Examples.

CALL ATOMIC_ADD(I[3], 42) causes the value of I on image 3 to become its previous value plus 42.

CALL ATOMIC_ADD(M[4], N, ORIG) causes the value of M on image 4 to become its previous value plus the value of N on this image. ORIG is defined with 99 if the previous value of M was 99 on image 4.

7.4.2 ATOMIC_AND (ATOM, VALUE) or ATOMIC_AND (ATOM, VALUE, OLD)

Description. Atomic bitwise AND operation.

Class. Atomic subroutine.

Arguments.

ATOM	shall be a scalar coarray or coindexed object and of type integer with kind ATOMIC_INT_KIND, where ATOMIC_INT_KIND is a named constant in the intrinsic module ISO_FORTRAN_ENV. It is an INTENT (INOUT) argument. ATOM becomes defined with the value IAND (ATOM, INT (VALUE, ATOMIC_INT_KIND)).
VALUE	shall be scalar and of type integer. It is an INTENT(IN) argument.
OLD	shall be a scalar and of the same type and kind as ATOM. It is an INTENT (OUT) argument. It is defined with the value of ATOM that was used for performing the bitwise AND operation.

Example. CALL ATOMIC_AND (I[3], 6, IOLD) causes I on image 3 to become defined with the value 4 and the value of IOLD on the image executing the statement to be defined with the value 5 if the value of I[3] was 5 when the bitwise AND operation executed.

7.4.3 ATOMIC_CAS (ATOM, OLD, COMPARE, NEW)

Description. Atomic compare and swap.

Class. Atomic subroutine.

Arguments.

ATOM	shall be a scalar coarray or coindexed object and of type integer with kind ATOMIC_INT_KIND or of type logical with kind ATOMIC_LOGICAL_KIND, where ATOMIC_INT_KIND and ATOMIC_LOGICAL_KIND are named constants in the intrinsic module ISO_FORTRAN_ENV. It is an INTENT (INOUT) argument. If the value of ATOM is equal to the value of COMPARE, ATOM becomes defined with the value of INT (NEW, ATOMIC_INT_KIND) if it is of type integer, and with the value of NEW if it is of type logical. If the value of ATOM is not equal to the value of COMPARE, the value of ATOM is not changed.
OLD	shall be scalar and of the same type and kind as ATOM. It is an INTENT (OUT) argument. It is defined with the value of ATOM that was used for performing the compare operation.
COMPARE	shall be scalar and of the same type and kind as ATOM. It is an INTENT(IN) argument.
NEW	shall be scalar and of the same type as ATOM. It is an INTENT(IN) argument.

Example. CALL ATOMIC_CAS(I[3], OLD, Z, 1) causes I on image 3 to become defined with the value 1 if its value is that of Z, and OLD to be defined with the value of I on image 3 that was used for performing the compare and swap operation.

7.4.4 ATOMIC_OR (ATOM, VALUE) or ATOMIC_OR (ATOM, VALUE, OLD)

Description. Atomic bitwise OR operation.

Class. Atomic subroutine.

Arguments.

ATOM	shall be a scalar coarray or coindexed object and of type integer with kind ATOMIC_INT_KIND, where ATOMIC_INT_KIND is a named constant in the intrinsic module ISO_FORTRAN_ENV. It is an INTENT (INOUT) argument. ATOM becomes defined with the value IOR (ATOM, INT (VALUE, ATOMIC_INT_KIND)).
VALUE	shall be scalar and of type integer. It is an INTENT(IN) argument.
OLD	shall be a scalar and of the same type and kind as ATOM. It is an INTENT (OUT) argument. It is defined with the value of ATOM that was used for performing the bitwise OR operation.

Example. CALL ATOMIC_OR (I[3], 1, IOLD) causes I on image 3 to become defined with the value 3 and the value of IOLD on the image executing the statement to be defined with the value 2 if the value of I[3] was 2 when the bitwise OR operation executed.

7.4.5 ATOMIC_XOR (ATOM, VALUE) or ATOMIC_XOR (ATOM, VALUE, OLD)

Description. Atomic bitwise exclusive OR operation.

Class. Atomic subroutine.

Arguments.

ATOM	shall be a scalar coarray or coindexed object and of type integer with kind ATOMIC_INT_KIND, where ATOMIC_INT_KIND is a named constant in the intrinsic module ISO_FORTRAN_ENV. It is an INTENT (INOUT) argument. ATOM becomes defined with the value IEOR (ATOM, INT (VALUE, ATOMIC_INT_KIND)).
VALUE	shall be scalar and of type integer. It is an INTENT(IN) argument.
OLD	shall be a scalar and of the same type and kind as ATOM. It is an INTENT (OUT) argument. It is defined with the value of ATOM that was used for performing the bitwise exclusive OR operation.

Example. CALL ATOMIC_XOR (I[3], 1, IOLD) causes I on image 3 to become defined with the value 2 and the value of IOLD on the image executing the statement to be defined with the value 3 if the value of I[3] was 3 when the bitwise exclusive OR operation executed.

7.4.6 CO_BROADCAST (SOURCE, SOURCE_IMAGE [, STAT, ERRMSG])

Description. Copy a value to all images of the current team.

Class. Collective subroutine.

Arguments.

SOURCE	shall have the same type and type parameters on all images of the current team. If it is an array, it shall have the same shape on all images of the current team. SOURCE becomes defined, as if by intrinsic assignment, on all images of the current team with the value of SOURCE on image SOURCE_IMAGE.
SOURCE_IMAGE	shall be a scalar of type integer. It is an INTENT(IN) argument. It shall be the image index of an image of the current team and have the same value on all images of the current team.
STAT (optional)	shall be a scalar of type default integer. It is an INTENT(OUT) argument.
ERRMSG (optional)	shall be a scalar of type default character. It is an INTENT(INOUT) argument.

The effect of the presence of the optional arguments STAT and ERRMSG is described in 7.3.

Example. If SOURCE is the array [1, 5, 3] on image one, after execution of CALL CO_BROADCAST(SOURCE,1) the value of SOURCE on all images of the current team is [1, 5, 3].

7.4.7 CO_MAX (SOURCE [, RESULT, RESULT_IMAGE, STAT, ERRMSG])

Description. Compute elemental maximum value on the current team of images.

Class. Collective subroutine.

Arguments.

SOURCE	shall be of type integer, real, or character. It shall have the same type and type parameters on all images of the current team. If it is a scalar, the computed value is equal to the maximum value of SOURCE on all images of the current team. If it is an array it shall have the same shape on all images of the current team and each element of the computed value is equal to the maximum value of all the corresponding elements of SOURCE on the images of the current team.
RESULT (optional)	shall be of the same type, type parameters, and shape as SOURCE. It is an INTENT(OUT) argument. If RESULT is present it shall be present on all images of the current team.
RESULT_IMAGE (optional)	shall be a scalar of type integer. It is an INTENT(IN) argument. If it is present, it

shall be present on all images of the current team, have the same value on all images of the current team, and that value shall be the image index of an image of the current team.

STAT (optional) shall be a scalar of type default integer. It is an INTENT(OUT) argument.

ERRMSG (optional) shall be a scalar of type default character. It is an INTENT(INOUT) argument.

If RESULT and RESULT_IMAGE are not present, the computed value is assigned to SOURCE on all the images of the current team. If RESULT is not present and RESULT_IMAGE is present, the computed value is assigned to SOURCE on image RESULT_IMAGE and SOURCE on all other images of the current team becomes undefined. If RESULT is present and RESULT_IMAGE is not present, the computed value is assigned to RESULT on all images of the current team. If RESULT and RESULT_IMAGE are present, the computed value is assigned to RESULT on image RESULT_IMAGE and RESULT on all other images of the current team becomes undefined. If RESULT is present, SOURCE is not modified.

The effect of the presence of the optional arguments STAT and ERRMSG is described in 7.3.

Example. If the number of images in the current team is two and SOURCE is the array [1, 5, 3] on one image and [4, 1, 6] on the other image, the value of RESULT after executing the statement CALL CO_MAX(SOURCE, RESULT) is [4, 5, 6] on both images.

7.4.8 CO_MIN (SOURCE [, RESULT, RESULT_IMAGE, STAT, ERRMSG])

Description. Compute elemental minimum value on the current team of images.

Class. Collective subroutine.

Arguments.

SOURCE shall be of type integer, real, or character. It shall have the same type and type parameters on all images of the current team. If it is a scalar, the computed value is equal to the minimum value of SOURCE on all images of the current team. If it is an array it shall have the same shape on all images of the current team and each element of the computed value is equal to the minimum value of all the corresponding elements of SOURCE on the images of the current team.

RESULT (optional) shall be of the same type, type parameters, and shape as SOURCE. It is an INTENT(OUT) argument. If RESULT is present it shall be present on all images of the current team.

RESULT_IMAGE (optional) shall be a scalar of type integer. It is an INTENT(IN) argument. If it is present, it shall be present on all images of the current team, have the same value on all images of the current team, and that value shall be the image index of an image of the current team.

STAT (optional) shall be a scalar of type default integer. It is an INTENT(OUT) argument.

ERRMSG (optional) shall be a scalar of type default character. It is an INTENT(INOUT) argument.

If RESULT and RESULT_IMAGE are not present, the computed value is assigned to SOURCE on all the images of the current team. If RESULT is not present and RESULT_IMAGE is present, the computed value is assigned to SOURCE on image RESULT_IMAGE and SOURCE on all other images of the current team becomes undefined. If RESULT is present and RESULT_IMAGE is not present, the computed value is assigned to RESULT on all images of the current team. If RESULT and RESULT_IMAGE are present, the computed value is assigned to RESULT on image RESULT_IMAGE and RESULT on all other images of the current team becomes undefined. If RESULT is present, SOURCE is not modified.

The effect of the presence of the optional arguments STAT and ERRMSG is described in 7.3.

Example. If the number of images in the current team is two and SOURCE is the array [1, 5, 3] on one image and [4, 1, 6] on the other image, the value of RESULT after executing the statement CALL CO_MIN(SOURCE, RESULT) is [1, 1, 3] on both images.

7.4.9 CO_REDUCE (SOURCE, OPERATOR [, RESULT, RESULT_IMAGE, STAT, ERRMSG])

Description. General reduction of elements on the current team of images.

Class. Collective subroutine.

Arguments.

SOURCE shall not be polymorphic. It shall have the same type and type parameters on all images of the current team. If SOURCE is a scalar, the computed value is the result of the reduction operation of applying OPERATOR to the values of SOURCE on all images of the current team. If SOURCE is an array it shall have the same shape on all images of the current team and each element of the computed value is equal to the result of the reduction operation of applying OPERATOR to all the corresponding elements of SOURCE on all the images of the current team.

OPERATOR shall be a pure function with two arguments of the same type and type parameters as SOURCE. Its result shall have the same type and type parameters as SOURCE. The arguments and result shall not be polymorphic. OPERATOR shall implement a mathematically commutative and associative operation. OPERATOR shall implement the same function on all images of the current team.

RESULT (optional) shall not be polymorphic. It shall be of the same type, type parameters, and shape as SOURCE. It is an INTENT(OUT) argument. If RESULT is present it shall be present on all images of the current team.

RESULT_IMAGE (optional) shall be a scalar of type integer. It is an INTENT(IN) argument. If it is present, it shall be present on all images of the current team, have the same value on all images of the current team, and that value shall be the image index of an image of the current team.

STAT (optional) shall be a scalar of type default integer. It is an INTENT(OUT) argument.

ERRMSG (optional) shall be a scalar of type default character. It is an INTENT(INOUT) argument.

If RESULT and RESULT_IMAGE are not present, the computed value is assigned to SOURCE on all images of the current team. If RESULT is not present and RESULT_IMAGE is present, the computed value is assigned to SOURCE on image RESULT_IMAGE and SOURCE on all other images of the current team becomes undefined. If RESULT is present and RESULT_IMAGE is not present, the computed value is assigned to RESULT on all images of the current team. If RESULT and RESULT_IMAGE are present, the computed value is assigned to RESULT on image RESULT_IMAGE and RESULT on all other images of the current team becomes undefined. If RESULT is present, SOURCE is not modified.

The computed value of a reduction operation over a set of values is the result of an iterative process. Each iteration involves the execution of `r = OPERATOR(x,y)` for `x` and `y` in the set, the removal of `x` and `y` from the set, and the addition of `r` to the set. The process terminates when the set has only one element which is the value of the reduction.

The effect of the presence of the optional arguments STAT and ERRMSG is described in 7.3.

Example. If the number of images in the current team is two and SOURCE is the array [1, 5, 3] on one image and [4, 1, 6] on the other image, and MyADD is a function that returns the sum of its two integer arguments, the value of RESULT after executing the statement CALL CO_REDUCE(SOURCE, MyADD, RESULT) is [5, 6, 9] on both images.

7.4.10 CO_SUM (SOURCE [, RESULT, RESULT_IMAGE, STAT, ERRMSG])

Description. Sum elements on the current team of images.

Class. Collective subroutine.

Arguments.

SOURCE shall be of numeric type. It shall have the same type and type parameters on all images of the current team. If it is a scalar, the computed value is equal to a processor-dependent and image-dependent approximation to the sum of the values of SOURCE on all images of the current team. If it is an array it shall have the same shape on all images of the current team and each element of the computed value is equal to a processor-dependent and image-dependent approximation to the sum of all the corresponding elements of SOURCE on the images of the current team.

RESULT (optional) shall be of the same type, type parameters, and shape as SOURCE. It is an INTENT(OUT) argument. If RESULT is present it shall be present on all images of the current team.

RESULT_IMAGE (optional) shall be a scalar of type integer. It is an INTENT(IN) argument. If it is present, it shall be present on all images of the current team, have the same value on all images of the current team, and that value shall be the image index of an image of the current team.

STAT (optional) shall be a scalar of type default integer. It is an INTENT(OUT) argument.

ERRMSG (optional) shall be a scalar of type default character. It is an INTENT(INOUT) argument.

If RESULT and RESULT_IMAGE are not present, the computed value is assigned to SOURCE on all the images of the current team. If RESULT is not present and RESULT_IMAGE is present, the computed value is assigned to SOURCE on image RESULT_IMAGE and SOURCE on all other images of the current team becomes undefined. If RESULT is present and RESULT_IMAGE is not present, the computed value is assigned to RESULT on all images of the current team. If RESULT and RESULT_IMAGE are present, the computed value is assigned to RESULT on image RESULT_IMAGE and RESULT on all other images of the current team becomes undefined. If RESULT is present, SOURCE is not modified.

The effect of the presence of the optional arguments STAT and ERRMSG is described in 7.3.

Example. If the number of images in the current team is two and SOURCE is the array [1, 5, 3] on one image and [4, 1, 6] on the other image, the value of RESULT after executing the statement CALL CO_SUM(SOURCE, RESULT) is [5, 6, 9] on both images.

7.4.11 EVENT_QUERY (EVENT, COUNT [, STAT, ERRMSG])

Description. Query the count of an event variable.

Class. Subroutine.

Arguments.

EVENT shall be scalar and of type EVENT_TYPE defined in the ISO_FORTRAN_ENV intrinsic module. It is an INTENT(IN) argument.

COUNT shall be scalar and of type integer with a decimal range no smaller that that of default integer. It is an INTENT(OUT) argument. If no error conditions occurs, COUNT is assigned the value of the number of successful posts minus the number of successful waits for EVENT. Otherwise, it is assigned the value 0.

STAT (optional) shall be scalar and of type default integer. It is an INTENT(OUT) argument. It is assigned the value 0 if no error condition occurs and a processor-defined positive value if an error condition occurs.

ERRMSG (optional) shall be a scalar of type default character. It is an INTENT(INOUT) argument.

If the ERRMSG argument is present and an error condition occurs, the processor shall assign an explanatory message to the argument. If no such condition occurs, the processor shall not change the value of the argument.

Example. If EVENT is an event variable for which there have been no successful posts or waits, after the invocation

```
CALL EVENT_QUERY ( EVENT, COUNT )
```

the integer variable COUNT has the value 0. If there have been 10 successful posts to EVENT[2] and 2 successful waits without an UNTIL_COUNT specification, after the invocation

```
CALL EVENT_QUERY ( EVENT[2], COUNT )
```

COUNT has the value 8.

NOTE 7.3

Execution of EVENT_QUERY does not imply any synchronization.

7.4.12 FAILED_IMAGES ([TEAM, KIND])

Description. Indices of failed images.

Class. Transformational function.

Arguments.

TEAM (optional) shall be a scalar of the type TEAM_TYPE defined in the ISO_FORTRAN_ENV intrinsic module. Its value shall represent an ancestor team.

KIND (optional) shall be a scalar integer constant expression. Its value shall be the value of a kind type parameter for the type INTEGER. The range for integers of this kind shall be at least as large as for default integer.

Result Characteristics. Integer. If KIND is present, the kind type parameter is that specified by the value of KIND; otherwise, the kind type parameter is that of default integer type. The result is an array of rank one whose size is equal to the number of failed images in the specified team.

Result Value. If TEAM is present, its value specifies the team; otherwise, the team specified is the current team. The elements of the result are the values of the image indices of the failed images in the specified team, in numerically increasing order.

Examples. If image 3 is the only failed image in the current team, FAILED_IMAGES() has the value [3]. If there are no failed images in the current team, FAILED_IMAGES() is a zero-sized array.

7.4.13 GET_TEAM (TEAM_VAR [,DISTANCE])

Description. Define TEAM_VAR with team value.

Class. Subroutine.

Arguments.

TEAM_VAR shall be scalar and of type TEAM_TYPE defined in the ISO_FORTRAN_ENV intrinsic module. It is an INTENT(OUT) argument. It shall not be the team variable of the current team, nor of any of its ancestors.

DISTANCE (optional) shall be scalar nonnegative integer. It is an INTENT(IN) argument.

If DISTANCE is not present, TEAM_VAR is defined with the value of a team variable of the current team. If DISTANCE is present with a value less than or equal to the team distance between the current team and the initial team, TEAM_VAR is defined with the value of a team variable of the ancestor team at that distance. Otherwise TEAM_VAR is defined with the value of a team variable for the initial team.

Examples.

```
   USE,INTRINSIC :: ISO_FORTRAN_ENV
   TYPE(TEAM_TYPE) :: WORLD_TEAM, TEAM2

   ! Define a team variable representing the initial team
   CALL GET_TEAM(WORLD_TEAM)
   END
   SUBROUTINE TT (A)
   USE,INTRINSIC :: ISO_FORTRAN_ENV
   REAL A[*]
   TYPE(TEAM_TYPE) :: NEW_TEAM, PARENT_TEAM

   ... ! Form NEW_TEAM

   CALL GET_TEAM(PARENT_TEAM)

   CHANGE TEAM(NEW_TEAM)

      ! Reference image 1 in parent's team
      A [PARENT_TEAM :: 1] = 4.2

      ! Reference image 1 in current team
      A [1] = 9.0
   END TEAM
   END SUBROUTINE TT
```

7.4.14 TEAM_DEPTH( )

Description. Team depth for the current team.

Class. Transformational function.

Arguments. None.

Result Characteristics. Scalar default integer.

Result Value. The result of TEAM_DEPTH is an integer with a value equal to the team distance between the current team and the initial team.

Example.

```
PROGRAM TD
   USE,INTRINSIC :: ISO_FORTRAN_ENV
   INTEGER         :: I_TEAM_DEPTH
   TYPE(TEAM_TYPE) :: TEAM

   FORM TEAM(1, TEAM)
   CHANGE TEAM(TEAM)
     I_TEAM_DEPTH = TEAM_DEPTH()
   END TEAM
END
```

On completion of the CHANGE TEAM construct, I_TEAM_DEPTH has the value 1.

7.4.15 TEAM_ID ([DISTANCE])

Description. Team identifier.

Class. Transformational function.

Argument. DISTANCE (optional) shall be a scalar nonnegative integer.

Result Characteristics. Default integer scalar.

Result Value. If DISTANCE is not present, the result value is the team identifier of the invoking image in the current team. If DISTANCE is present with a value less than or equal to the team distance between the current team and the initial team, the result has the value of the team identifier that the invoking image had when it was a member of the team with a team distance of DISTANCE from the current team. Otherwise, the result has the value 1.

Example. The following code illustrates the use of TEAM_ID to control which code is executed.

```
TYPE(TEAM_TYPE) :: ODD_EVEN
    :
ME = THIS_IMAGE()
FORM TEAM ( 2-MOD(ME,2), ODD_EVEN )
CHANGE TEAM (ODD_EVEN)
  SELECT CASE (TEAM_ID())
  CASE (1)
     :  ! Code for odd images in parent team
  CASE (2)
     :  ! Code for even images in parent team
  END SELECT
END TEAM
```

7.5 Modified intrinsic procedures

7.5.1 MOVE_ALLOC

The description of the intrinsic function MOVE_ALLOC in ISO/IEC 1539-1:2010, as modified by ISO/IEC 1539-1:2010/Cor 2:2013, is changed to take account of the possibility of failed images and to add two optional arguments, STAT and ERRMSG, and a modified result if either is present.

The STAT argument shall be a scalar of type default integer. It is an INTENT(OUT) argument.

The ERRMSG argument shall be a scalar of type default character. It is an INTENT(INOUT) argument.

If the execution is successful

(1) The allocation status of TO becomes unallocated if FROM is unallocated on entry to MOVE_ALLOC. Otherwise, TO becomes allocated with dynamic type, type parameters, array bounds, array cobounds, and value identical to those that FROM had on entry to MOVE_ALLOC.
(2) If TO has the TARGET attribute, any pointer associated with FROM on entry to MOVE_ALLOC becomes correspondingly associated with TO. If TO does not have the TARGET attribute, the pointer association status of any pointer associated with FROM on entry becomes undefined.
(3) The allocation status of FROM becomes unallocated.

When a reference to MOVE_ALLOC is executed for which the FROM argument is a coarray, there is an implicit synchronization of all nonfailed images of the current team. On each nonfailed image, execution of the segment (8.5.2 of ISO/IEC 1539-1:2010) following the CALL statement is delayed until all other nonfailed images of the current team have executed the same statement the same number of times.

If the STAT argument is present and execution is successful, the argument is assigned the value zero.

If the STAT argument is present and an error condition occurs, the argument is assigned a nonzero value. The value shall be that of the constant STAT_FAILED_IMAGE in the intrinsic module ISO_FORTRAN_ENV if the reason is that a failed image has been detected in the current team; otherwise, the value shall be that of the constant STAT_STOPPED_IMAGE in the intrinsic module ISO_FORTRAN_ENV if the reason is that a successful execution would have involved an interaction with an image that has initiated termination; otherwise, the value is a processor-dependent positive value that is different from the value of STAT_STOPPED_IMAGE or STAT_FAILED_IMAGE.

If the ERRMSG argument is present and an error condition occurs, the processor shall assign an explanatory message to the argument. If no such condition occurs, the processor shall not change the value of the argument.

7.5.2 NUM_IMAGES

The description of the intrinsic function NUM_IMAGES in ISO/IEC 1539-1:2010 is changed by adding two optional arguments DISTANCE and FAILED and a modified result if either is present.

The DISTANCE argument shall be a nonnegative scalar integer. If DISTANCE is not present the team specified is the current team. If DISTANCE is present with a value less than or equal to the team distance between the current team and the initial team, the team specified is the team of which the invoking image was a member with a team distance of DISTANCE from the current team; otherwise, the team specified is the initial team.

The FAILED argument shall be a scalar of type LOGICAL. If FAILED is not present the result is the number of images in the team specified. If FAILED is present with the value true, the result is the number of failed images in the team specified, otherwise the result is the number of nonfailed images in the team specified.

7.5.3 THIS_IMAGE

The description of the intrinsic function THIS_IMAGE( ) in ISO/IEC 1539-1:2010 is changed by adding an optional argument DISTANCE and a modified result if DISTANCE is present.

The DISTANCE argument shall be a scalar integer. It shall be nonnegative. If DISTANCE is not present, the result value is the image index of the invoking image in the current team. If DISTANCE is present with a value less than or equal to the team distance between the current team and the initial team, the result has the value of the image index of the invoking image in the ancestor team with a team distance of DISTANCE from the current team; otherwise, the result has the value of the image index that the invoking image had in the initial team.

(Blank page)

8 Required editorial changes to ISO/IEC 1539-1:2010(E)

8.1 General

The following editorial changes, if implemented, would provide the facilities described in foregoing clauses of this Technical Specification. Descriptions of how and where to place the new material are enclosed in braces {}. Edits to different places within the same clause are separated by horizontal lines.

In the edits, except as specified otherwise by the editorial instructions, underwave (underwave) and strike-out (~~strike-out~~) are used to indicate insertion and deletion of text.

8.2 Edits to Introduction

Include clauses a needed.

{In paragraph 1 of the Introduction}

After "informally known as Fortran 2008, plus the facilities defined in ISO/IEC TS 29113:2012" add "and ISO/IEC TS 18508:2014".

{After paragraph 3 of the Introduction and after the paragraph added by ISO/IEC TS 29113:2012, insert new paragraph}

ISO/IEC TS 18508 provides additional facilities for parallel programming:

• teams provide a capability for a subset of the images of the program to act as if it consists of all images for the purposes of image index values, coarray allocations, and synchronization.

• collective subroutines perform computations based on values on all the images of the current team, offering the possibility of efficient execution of reduction operations;

• atomic memory operations provide powerful low-level primitives for synchronization of activities among images and performing limited remote computation;

• tagged events allow one-sided ordering of execution segments;

• features for the support of continued execution after one or more images have failed; and

• features to detect which images have failed and simulate failure of an image.

8.3 Edits to clause 1

{In 1.3 Terms and definitions, insert new terms as follows}

1.3.30a
collective subroutine
intrinsic subroutine that is invoked on the current team of images to perform a calculation on those images and assign the computed value on one or all of them (13.1)

1.3.85a
failed image
an image for which references or definitions of a variable on the image fail when that variable should be accessible, or the image fails to respond during the execution of an image control statement or a reference to a collective

subroutine (13.8.2.21b)

1.3.145a
team
set of images that can readily execute independently of other images (2.3.4)

1.3.145a.1
current team
the team specified in the CHANGE TEAM statement of the innermost executing CHANGE TEAM construct, or the initial team if no CHANGE TEAM construct is active (2.3.4)

1.3.145a.2
initial team
the current team when the program began execution (2.3.4)

1.3.145a.3
parent team
team from which the current team was formed by executing a FORM TEAM statement (2.3.4)

1.3.145a.4
team identifier
integer value identifying a team (2.3.4)

1.3.145a.5
team distance
the distance between a team and one of its ancestors (2.3.4)

1.3.154.1-
event variable
scalar variable of type EVENT_TYPE (13.8.2.8a) from the intrinsic module ISO_FORTRAN_ENV

1.3.154.3
team variable
scalar variable of type TEAM_TYPE (13.8.2.26) from the intrinsic module ISO_FORTRAN_ENV

8.4 Edits to clause 2

{In 2.1 High level syntax, Add new construct and statements into the syntax list as follows: In R213 *executable-construct* insert alphabetically "*change-team-construct*"; in R214 *action-stmt* insert alphabetically "*event-post-stmt*", "*event-wait-stmt*", "*fail-image-stmt*", "*form-team-stmt*", and "*sync-team-stmt*".

{In 2.3.4 Program execution, after the first paragraph, insert 5.1, paragraphs 1 through 3, of this Technical Specification with the following changes: In the first paragraph delete "in ISO/IEC 1539-1:2010" following "R624" and insert "(8.5.2c)" following "FORM TEAM statement". In the third paragraph insert "(8.1.4a)" following "CHANGE TEAM construct". }

{In 2.4.7 Coarray, after the first paragraph, insert 5.1 paragraph 4 of this Technical Specification.}

{In 2.4.7 Coarray, edit the second paragraph as follows.}

For each coarray on an image of a team, there is a corresponding coarray with the same type, type parameters, and bounds on every other image of that team.

{In 2.4.7 Coarray, edit the first sentence of the third paragraph as follows.}

The set of corresponding coarrays on all images of a team is arranged in a rectangular pattern.

{In 2.4.7 Coarray, edit the first sentence of the fourth paragraph as follows.}

A coarray on any image of the current team can be accessed directly by using cosubscripts.

8.5 Edits to clause 4

{In 4.5.6.2 The finalization process, add to the end of NOTE 4.48}

in the current team

8.6 Edits to clause 6

{In 6.6 Image selectors, replace R624 with}

R624 *image-selector* **is** *lbracket* [*team-variable* ::] *cosubscript-list* ■
■ [, TEAM_ID = *scalar-int-expr*] *rbracket*

C627a (R624) *team-variable* and TEAM_ID = shall not both appear in the same *image-selector*.

{In 6.6 Image selectors, edit the last sentence of the second paragraph as follows.}

An image selector shall specify an image index value that is not greater than the number of images in the team specified by *team-variable* or a TEAM_ID specifier if either appears or in the current team otherwise.

{In 6.6 Image selectors, after paragraph 2 insert the two paragraphs following C508 in 5.4 of this Technical Specification with the following change: following "FORM TEAM statement" insert "(8.5.2c)" }

{In 6.7.1.2, Execution of an ALLOCATE statement, edit paragraphs 3 and 4 as follows}

If an *allocation* specifies a coarray, its dynamic type and the values of corresponding type parameters shall be the same on every image in the current team. The values of corresponding bounds and corresponding cobounds shall be the same on ~~every image~~ these images. If the coarray is a dummy argument, its ultimate argument (12.5.2.3) shall be the same coarray on ~~every image~~ these images.

When an ALLOCATE statement is executed for which an *allocate-object* is a coarray, there is an implicit synchronization of all nonfailed images in the current team. On ~~each image~~ these images, execution of the segment (8.5.2) following the statement is delayed until all other nonfailed images in the current team have executed the same statement the same number of times.

{In 6.7.3.2, Deallocation of allocatable variables, edit paragraphs 11 and 12 as follows}

When a DEALLOCATE statement is executed for which an *allocate-object* is a coarray, there is an implicit synchronization of all nonfailed images in the current team. On ~~each image~~ these images, execution of the segment (8.5.2) following the statement is delayed until all other nonfailed images in the current team have executed the same statement the same number of times. If the coarray is a dummy argument, its ultimate argument (12.5.2.3) shall be the same coarray on ~~every image~~ these images.

There is also an implicit synchronization of all nonfailed images in the current team in association with the deallocation of a coarray or coarray subcomponent caused by the execution of a RETURN or END statement or the termination of a BLOCK construct.

{In 6.7.4 STAT= specifier, para 3, replace the text to the bullet list with}

If the STAT= specifier appears in an ALLOCATE or DEALLOCATE statement with a coarray *allocate-object* and an error condition occurs, the specified variable is assigned a positive value. The value shall be that of the constant STAT_FAILED_IMAGE in the intrinsic module ISO_FORTRAN_ENV (13.8.2) if the reason is that a failed image has been detected in the current team; otherwise, the value shall be that of the constant STAT_STOPPED_-

IMAGE in the intrinsic module ISO_FORTRAN_ENV (13.8.2) if the reason is that a successful execution would have involved an interaction with an image that has initiated termination; otherwise, the value is a processor-dependent positive value that is different from the value of STAT_STOPPED_IMAGE or STAT_FAILED_IMAGE in the intrinsic module ISO_FORTRAN_ENV (13.8.2). In all of these cases, each *allocate-object* has a processor-dependent status.

8.7 Edits to clause 8

{In 8.1.1 General, paragraph 1, following the BLOCK construct entry in the list of constructs insert}

- CHANGE TEAM construct;

{Following 8.1.4 BLOCK construct insert 5.3 CHANGE TEAM construct from this Technical Specification as 8.1.4a, with rule, constraint, and Note numbers modified, the reference "(5.2)" in C506 changed to "(13.8.2.26)", and in the third paragraph following C506, delete "of ISO/IEC 1539-1:2010". }

{In 8.1.5 CRITICAL construct: In para 1, line 1, after "one image" add "of the current team". In para 3, line 1, after "other image" add "of the current team".}

{Following 8.4 STOP and ERROR STOP statements, insert 5.6 FAIL IMAGE statement from this Technical Specification as 8.4a, with rule and Note numbers modified.}

{In 8.5.1 Image control statements, paragraph 2, insert extra bullet points following the CRITICAL and END CRITICAL line}

- CHANGE TEAM and END TEAM;
- EVENT POST and EVENT WAIT;
- FORM TEAM;
- SYNC TEAM;

{In 8.5.1 Image control statements, edit paragraph 3 as follows}

All image control statements except CRITICAL, END CRITICAL, FORM TEAM, LOCK, and UNLOCK include the effect of executing a SYNC MEMORY statement (8.5.5).

{In 8.5.2 Segments, after the first sentence of paragraph 3, insert the following }

A coarray that is of type EVENT_TYPE may be referenced or defined during the execution of a segment that is unordered relative to the execution of another segment in which that coarray of type EVENT_TYPE is defined.

{Following 8.5.2 Segments insert 6.3 EVENT POST statement from this Technical Specification as 8.5.2a, with rule and constraint numbers modified, and change the "(6.2)" in C604 to "(13.8.2.8a)", and change the "(6.5)" at the end of the paragraph of text to "(13.8.2.21a)" }

{Following 8.5.2 Segments insert 6.4 EVENT WAIT statement from this Technical Specification as 8.5.2b, with rule and constraint numbers modified.}

{Following 8.5.2 Segments insert 5.4 FORM TEAM statement from this Technical Specification as 8.5.2c, with rule and Note numbers modified.}

{In 8.5.3 SYNC ALL statement, edit paragraph 2 as follows}

Execution of a SYNC ALL statement performs a synchronization of all nonfailed images in the current team.

Execution on an image, M, of the segment following the SYNC ALL statement is delayed until each other nonfailed image in the team has executed a SYNC ALL statement as many times as has image M. The segments that executed before the SYNC ALL statement on an image precede the segments that execute after the SYNC ALL statement on another image.

{In 8.5.4 SYNC IMAGES, edit paragraphs 1 through 3 as follows}

If *image-set* is an array expression, the value of each element shall be positive and not greater than the number of images in the current team, and there shall be no repeated values.

If *image-set* is a scalar expression, its value shall be positive and not greater than the number of images in the current team.

An *image-set* that is an asterisk specifies all images in the current team.

{Following 8.5.5 SYNC MEMORY statement, insert 5.5 SYNC TEAM statement from this Technical Specification as 8.5.5a, with the rule number modified.}

{In 8.5.7 STAT= and ERRMSG= specifiers in image control statements replace paragraphs 1 and 2 by}

If the STAT= specifier appears in a CHANGE TEAM, END TEAM, EVENT POST, EVENT WAIT, FORM TEAM, LOCK, SYNC ALL, SYNC IMAGES, SYNC MEMORY, SYNC TEAM, or UNLOCK statement and its execution is successful, the specified variable is assigned the value zero.

If the STAT= specifier appears in a CHANGE TEAM, END TEAM, EVENT POST, EVENT WAIT, FORM TEAM, LOCK, SYNC ALL, SYNC IMAGES, SYNC MEMORY, SYNC TEAM, or UNLOCK statement and an error condition occurs, the specified variable is assigned a positive value. The value shall be the constant STAT_FAILED_IMAGE in the intrinsic module ISO_FORTRAN_ENV (13.8.2) if the reason is that a failed image has been detected in the current team; otherwise, the value shall be the constant STAT_STOPPED_IMAGE in the intrinsic module ISO_FORTRAN_ENV (13.8.2) if the reason is that a successful execution would have involved an interaction with an image that has initiated termination; otherwise, the value is a processor-dependent positive value that is different from the value of STAT_STOPPED_IMAGE or STAT_FAILED_IMAGE in the intrinsic module ISO_FORTRAN_ENV (13.8.2).

If the STAT= specifier appears in a CHANGE TEAM, END TEAM, EVENT POST, EVENT WAIT, SYNC ALL, SYNC IMAGES, or SYNC TEAM statement and an error condition other than detection of a failed image occurs, the effect is the same as that of executing the SYNC MEMORY statement, except for defining the STAT= variable.

{In 8.5.7 STAT= and ERRMSG= specifiers in image control statements replace paragraphs 4 and 5 by}

If the STAT= specifier does not appear in a CHANGE TEAM, END TEAM, EVENT POST, EVENT WAIT, FORM TEAM, LOCK, SYNC ALL, SYNC IMAGES, SYNC MEMORY, SYNC TEAM, or UNLOCK statement and its execution is not successful, error termination is initiated.

If an ERRMSG= specifier appears in a CHANGE TEAM, END TEAM, EVENT POST, EVENT WAIT, FORM TEAM, LOCK, SYNC ALL, SYNC IMAGES, SYNC MEMORY, SYNC TEAM, or UNLOCK statement and its execution is not successful, the processor shall assign an explanatory message to the specified variable. If the execution is successful, the processor shall not change the value of the variable.

8.8 Edits to clause 13

{In 13.1 Classes of intrinsic procedures, edit paragraph 1 as follows}

Intrinsic procedures are divided into ~~seven~~ eight classes: inquiry functions, elemental functions, transformational functions, elemental subroutines, pure subroutines, atomic subroutines, collective subroutines, and (impure) subroutines.

{In 13.1 Classes of intrinsic procedures, append the following text to the end of paragraph 3}

For invocation of an atomic subroutine with an argument OLD, the assignment of the value to OLD is not part of the atomic action. For invocation of an atomic subroutine, evaluation of an INTENT(IN) argument is not part of the atomic action. If two or more variables are updated by a sequence or atomic memory operations on an image, and these changes are observed by atomic accesses from an unordered segment on another image, the changes need not be observed on the remote image in the same order as they are made on the local image, even if the updates in the local images are made in ordered segments.

{In 13.1 Classes of intrinsic procedures, insert the contents of 7.3 Collective subroutines of this Technical Specification after paragraph 3 and Note 13.1, with these changes: Paragraph 2 of 7.3. Delete "of ISO/IEC 1539-1:2010" Paragraph 5 of 7.3. Add "(13.8.2)" after "ISO_FORTRAN_ENV" twice.}

{In 13.5 Standard generic intrinsic procedures, paragraph 2 after the line "A indicates ... atomic subroutine" insert a new line}

C indicates that the procedure is a collective subroutine

{In 13.5 Standard generic intrinsic procedures, Table 13.1, insert new entries into the table, alphabetically}

ATOMIC_ADD	(ATOM, VALUE) or (ATOM, VALUE, OLD)	A	Atomic ADD operation.
ATOMIC_AND	(ATOM, VALUE) or (ATOM, VALUE, OLD)	A	Atomic bitwise AND operation.
ATOMIC_CAS	(ATOM, OLD, COMPARE, NEW)	A	Atomic compare and swap.
ATOMIC_OR	(ATOM, VALUE) or (ATOM, VALUE, OLD)	A	Atomic bitwise OR operation.
ATOMIC_XOR	(ATOM, VALUE) or (ATOM, VALUE, OLD)	A	Atomic bitwise exclusive OR operation.
CO_BROADCAST	(SOURCE, SOURCE_IMAGE)	C	Copy a value to all images of the current team.
CO_MAX	(SOURCE [, RESULT, RESULT_IMAGE])	C	Compute maximum of elements across images.
CO_MIN	(SOURCE [, RESULT, RESULT_IMAGE])	C	Compute minimum of elements across images.
CO_REDUCE	(SOURCE, OPERATOR [, RESULT, RESULT_IMAGE])	C	General reduction of elements across images.
CO_SUM	(SOURCE [, RESULT, RESULT_IMAGE])	C	Sum elements across images.
EVENT_QUERY	(EVENT, COUNT[, STATUS])	S	Count of an event.
FAILED_IMAGES	([TEAM, KIND])	T	Indices of failed images.
GET_TEAM	(TEAM_VAR [, DISTANCE])	S	Define TEAM_VAR with team value.
TEAM_DEPTH	( )	T	Team depth for this image.
TEAM_ID	([DISTANCE])	T	Team identifier.

{In 13.5 Standard generic intrinsic procedures, Table 13.1, edit the entries for MOVE_ALLOC, NUM_IMAGES, and THIS_IMAGE as follows}

MOVE_ALLOC	(FROM, TO [, STAT, ERRMSG])	PS	Move an allocation.
NUM_IMAGES	([DISTANCE, FAILED])	T	Number of images.
THIS_IMAGE	([DISTANCE])	T	Index of the invoking image.

{In 13.7 Specifications of the standard intrinsic procedures, insert subclauses 7.4.1 through 7.4.15 of this Technical Specification in order alphabetically, with subclause numbers adjusted accordingly.}

{In 13.7.118 MOVE_ALLOC, edit the subclause title as follows}

13.7.118 MOVE_ALLOC (FROM, TO [, STAT, ERRMSG])

{In 13.7.118 MOVE_ALLOC, add the arguments descriptions as follows}

STAT (optional) shall be a scalar of type default integer. It is an INTENT(OUT) argument.

ERRMSG (optional) shall be a scalar of type default character. It is an INTENT(INOUT) argument.

{In 13.7.118 MOVE_ALLOC, replace paragraphs 4 through 6 and the paragraph that was added by ISO/IEC 1539-1:2010/Cor 2:2013 by paragraphs 4 through 8 of 7.5.1 of this Technical Specification, deleting "of ISO/IEC 1539-1:2010" in paragraph 5.}

{In 13.7.126 NUM_IMAGES, edit the subclause title as follows}

13.7.126 NUM_IMAGES ([DISTANCE, FAILED])

{In 13.7.126 NUM_IMAGES, replace paragraph 3 with}

Arguments.

DISTANCE (optional) shall be a nonnegative scalar integer. It is an INTENT(IN) argument.

FAILED (optional) shall be a scalar of type LOGICAL. Its value determines whether the result is the number of failed images or the number of nonfailed images. It is an INTENT(IN) argument.

{In 13.7.126 NUM_IMAGES, replace paragraph 5 with}

Result Value.

If DISTANCE is not present, the team specified is the current team. If DISTANCE is present with a value less than or equal to the team distance between the current team and the initial team, the team specified is the team of which the invoking image was a member with a team distance of DISTANCE from the current team; otherwise, the team specified is the initial team.

If FAILED is not present, the result is the number of images in the team specified. If FAILED is present with the value true, the result is the number of failed images in the team specified; otherwise, the result is the number of nonfailed images in the team specified.

{In 13.7.165 THIS_IMAGE ( ) or THIS_IMAGE (COARRAY [, DIM]) edit the subclause title as follows }

13.7.165 THIS IMAGE ([DISTANCE]) or THIS IMAGE (COARRAY [, DIM])

{In 13.7.165 THIS_IMAGE ( ) or THIS_IMAGE (COARRAY [, DIM]) insert a new argument at the end of paragraph 3 }

DISTANCE (optional) shall be a scalar integer. It shall be nonnegative. It shall not be a coarray.

{In 13.7.165 THIS_IMAGE ( ) or THIS_IMAGE (COARRAY [, DIM]) replace *Case(i):* in paragraph 5 with }

Case (i): If DISTANCE is not present the result value is the image index of the invoking image in the current team. If DISTANCE is present with a value less than or equal to the team distance between the current team and the initial team, the result has the value of the image index in the team of which the invoking image was a member with a team distance of DISTANCE from the current team; otherwise, the result has the value of the image index that the invoking image had in the initial team.

{In 13.7.172 UCOBOUND, edit the Result Value as follows.}

The final upper cobound is the final cosubscript in the cosubscript list for the coarray that selects the image with index ~~NUM_IMAGES( )~~ equal to the number of images in the current team when the coarray was established.

{In 13.8.2 The ISO_FORTRAN_ENV intrinsic module, insert a new subclause 13.8.2.8a consisting of subclause 6.2 EVENT_TYPE of this Technical Specification, but omitting the final sentence of the first paragraph and the fourth sentence of the second paragraph.}

{In 13.8.2 The ISO_FORTRAN_ENV intrinsic module, insert a new subclause 13.8.2.21b consisting of subclause 5.6 STAT_FAILED_IMAGE of this Technical Specification, but omitting the final sentence of the paragraph.}

{In 13.8.2 The ISO_FORTRAN_ENV intrinsic module, append a new subclause 13.8.2.26 consisting of subclause 5.2 TEAM_TYPE of this Technical Specification, but omitting the final sentence of the first paragraph.}

8.9 Edits to clause 16

{In 16.4 Statement and construct entities, add the following new paragraph after paragraph 8}

The associate names of a CHANGE TEAM construct have the scope of the block. They have the declared type, dynamic type, type parameters, rank, and bounds of the corresponding coselector.

{In 16.5.1.6 Construct association, append the following sentence to the paragraph 1}

Execution of a CHANGE TEAM statement establishes an association between each coselector and the corresponding associate name of the construct.

{At the end of the list of variable definition contexts in 16.6.7 para 1, replace the "." at the end of entry (15) with ";" and add two new entries as follows}

(16) a *team-variable* in a FORM TEAM statement;

(17) an *event-variable* in an EVENT POST or EVENT WAIT statement.

8.10 Edits to annex A

{At the end of A.2 Processor dependencies, replace the final full stop with a semicolon and add new items as follows}

- the conditions that cause an image to fail;
- the manner in which the stop code of the FAIL IMAGE statement is made available;
- the computed value of the CO_SUM intrinsic subroutine;
- the computed value of the CO_REDUCE intrinsic subroutine;
- how sequences of event posts in unordered segments interleave with each other;
- the image index value assigned by a FORM TEAM statement without a NEW_INDEX= specifier.

8.11 Edits to annex C

{In C.5 Clause 8 notes, at the end of the subclause insert subcauses A.1.1, A.1.2, A.1.3, A.1.4, A.2.1, and A.2.2 from this Technical Specification as subclauses C.5.5 to C.5.10.}

{In C.10 Clause 13 notes, at the end of the subclause insert subcauses A.3.1 and A.3.2 from this Technical Specification as subclauses C.10.2 and C.10.3.}

Annex A

(Informative)

Extended notes

A.1 Clause 5 notes

A.1.1 Example using three teams

Compute fluxes over land, sea and ice in different teams based on surface properties. Assumption: Each image deals with areas containing exactly one of the three surface types.

```
SUBROUTINE COMPUTE_FLUXES(FLUX_MOM, FLUX_SENS, FLUX_LAT)
USE,INTRINSIC :: ISO_FORTRAN_ENV
REAL, INTENT(OUT) :: FLUX_MOM(:,:), FLUX_SENS(:,:), FLUX_LAT(:,:)
INTEGER, PARAMETER :: LAND=1, SEA=2, ICE=3
CHARACTER(LEN=10)  :: SURFACE_TYPE
INTEGER            :: MY_SURFACE_TYPE, N_IMAGE
TYPE(TEAM_TYPE)    :: TEAM_SURFACE_TYPE

   CALL GET_SURFACE_TYPE(THIS_IMAGE(), SURFACE_TYPE) ! Surface type
   SELECT CASE (SURFACE_TYPE)                   ! of the executing image
   CASE ('LAND')
      MY_SURFACE_TYPE = LAND
   CASE ('SEA')
      MY_SURFACE_TYPE = SEA
   CASE ('ICE')
      MY_SURFACE_TYPE = ICE
   CASE DEFAULT
      ERROR STOP
   END SELECT
   FORM TEAM(MY_SURFACE_TYPE, TEAM_SURFACE_TYPE)

   CHANGE TEAM(TEAM_SURFACE_TYPE)
      SELECT CASE (TEAM_ID( ))
      CASE (LAND    )   ! Compute fluxes over land surface
         CALL COMPUTE_FLUXES_LAND(FLUX_MOM, FLUX_SENS, FLUX_LAT)
      CASE (SEA)    ! Compute fluxes over sea surface
         CALL COMPUTE_FLUXES_SEA(FLUX_MOM, FLUX_SENS, FLUX_LAT)
      CASE (ICE)    ! Compute fluxes over ice surface
         CALL COMPUTE_FLUXES_ICE(FLUX_MOM, FLUX_SENS, FLUX_LAT)
      CASE DEFAULT
         ERROR STOP
      END SELECT
   END TEAM
END SUBROUTINE COMPUTE_FLUXES
```

A.1.2 Example involving failed images

Parallel algorithms often use work sharing schemes based on a specific mapping between image indices and global data addressing. To allow such programs to continue when one or more images fail, spare images can be used to re-establish execution of the algorithm with the failed images replaced by spare images, while retaining the image mapping.

The following example illustrates how this might be done. In this setup, failure cannot be tolerated for image 1 and the spare images, whose number is assumed to be small compared to the number of active images.

```
PROGRAM possibly_recoverable_simulation
  USE, INTRINSIC :: iso_fortran_env
  IMPLICIT NONE
  INTEGER, ALLOCATABLE :: failed_img(:)
  INTEGER :: images_used, i, images_spare, id, me, status
  TYPE(team_type) :: simulation_team
  LOGICAL :: read_checkpoint, done[*]

  images_used = ...  ! A value slightly less num_images()
  images_spare = num_images() - images_used
  read_checkpoint = this_image() > images_used

  setup : DO
    me = this_image()
    id = 1
    IF (me > images_used) id = 2
!
!   set up spare images as replacement for failed ones
    failed_img = failed_images()
    if (size(failed_img) > images_spare) ERROR STOP 'cannot recover'
    DO i=1, size(failed_img)
       IF (failed_img(i) > images_used .or. &
           failed_img(i) == 1)  ERROR STOP 'cannot recover'
       IF (me == images_used + i) THEN
           me = failed_img(i)
           id = 1
       END IF
    END DO
!
!   set up a simulation team of constant size.
!   id == 2 does not participate in team execution
    FORM TEAM (id, simulation_team, NEW_INDEX=me, STAT=status)
    simulation : CHANGE TEAM (simulation_team, STAT=status)
      IF (TEAM_ID() == 1) THEN
        iter : DO
          CALL simulation_procedure(read_checkpoint, status, done)
!         simulation_procedure:
!           sets up required objects (maybe coarrays)
!           reads checkpoint if requested
!           returns status on its internal synchronizations
!           returns .TRUE. in done once complete
            read_checkpoint = .FALSE.
          IF (status == STAT_FAILED_IMAGE) THEN
             read_checkpoint = .TRUE.
             EXIT simulation
          ELSE IF (done)
             EXIT iter
          END IF
        END DO iter
      END IF
    END TEAM simulation (STAT=status)
    SYNC ALL (STAT=status)
    IF (this_image() > images_used) done = done[1]
    IF (done) EXIT setup
  END DO setup
END PROGRAM possibly_recoverable_simulation
```

Supporting fail-safe execution imposes obligations on library writers who use the parallel language facilities. Every synchronization statement, allocation or deallocation of coarrays, or invocation of a collective procedure must specify a synchronization status variable, and implicit deallocation of coarrays must be avoided. In particular, coarray module variables that are allocated inside the team execution context are not persistent.

A.1.3 Accessing coarrays in sibling teams

The following program shows the subdivision of a 4 x 4 grid into 2 x 2 teams and addressing of sibling teams.

```
PROGRAM DEMO
! Initial team : 16 images. Algorithm design is a 4 x 4 grid.
! Desire 4 teams, for the upper left (UL), upper right (UR),
!                          Lower left (LL), lower right (LR)
  USE,INTRINSIC :: ISO_FORTRAN_ENV, ONLY: team_type
  TYPE (team_type) :: t
  INTEGER,PARAMETER :: UL=11, UR=22, LL=33, LR=44
  REAL    :: A(10,10)[4,*]
  INTEGER :: mype, teamid, newpe
  INTEGER :: UL_image_list(4) = [1, 2, 5, 6], &
             LL_image_list(4) = UL_image_list + 2,  &
             UR_image_list(4) = UL_image_list + 8,  &
             LR_image_list(4) = UL_image_list + 10

  mype = THIS_IMAGE()
  IF (any(mype == UL_image_list)) teamid = UL
  IF (any(mype == LL_image_list)) teamid = LL
  IF (any(mype == UR_image_list)) teamid = UR
  IF (any(mype == LR_image_list)) teamid = LR
  FORM TEAM (teamid, t)

  a = 3.14

  CHANGE TEAM (t, b[2,*] => a)
    ! Inside change team, image pattern for B is a 2 x 2 grid
    b(5,5) = b(1,1)[2,1]

    ! Outside the team addressing:

    newpe = THIS_IMAGE()
    SELECT CASE (team_id())
    CASE (UL)
       IF (newpe == 3) THEN
           b(:,10) = b(:,1)[1, 1, TEAM_ID=UR]  ! Right column of UL gets
                                               ! left column of UR
       ELSE IF (newpe == 4) THEN
           b(:,10) = b(:,1)[2, 1, TEAM_ID=UR]
       END IF
    CASE (LL)
```

```
       ! Similar to complete column exchange across middle of the
       ! original grid
    END SELECT
   END TEAM
END PROGRAM DEMO
```

A.1.4 Reducing the codimension of a coarray

This example illustrates how to use a subroutine to coordinate cross-image access to a coarray for row and column processing.

```
PROGRAM row_column
  USE, INTRINSIC :: iso_fortran_env, ONLY : team_type
  IMPLICIT NONE

  TYPE(team_type), target :: row_team, col_team
  TYPE(team_type), pointer :: used_team
  REAL, ALLOCATABLE :: a(:,:)[:,:]
  INTEGER :: ip, na, p, me(2)

  p = ... ; q = ... ! such that p**q == num_images()
  na = ...          ! local problem size

  ! allocate and initialize data
  ALLOCATE(a(na,na)[p,*])
  a = ...

  me = this_image(a)

  FORM TEAM(me(1), row_team, NEW_INDEX=me(2))
  FORM TEAM(me(2), col_team, NEW_INDEX=me(1))

  ! make a decision on whether to process by row or column
  IF (...) THEN
     used_team => row_team
  ELSE
     used_team => col_team
  END IF

  ... ! do local computations on a

  CHANGE TEAM (used_team)

    CALL further_processing(a, ...)

  END TEAM
CONTAINS
  SUBROUTINE further_processing(a, ...)
    REAL :: a(:,:)[*]
    INTEGER :: ip

    ! update ip-th row or column submatrix
    a(:,:)[ip] = ...

    SYNC ALL
```

```
      ... ! do further local computations on a

    END SUBROUTINE
END PROGRAM row_column
```

A.2 Clause 6 notes

A.2.1 EVENT_QUERY example

The following example illustrates the use of events via a program whose first image shares out work items to all other images. Only one work item at a time can be active on the worker images, and these deal with the result (e.g. via I/O) without directly feeding data back to the master image.

Because the work items are not expected to be balanced, the master keeps cycling through all available images in order to find one that is waiting for work.

Furthermore, the master keeps track of failed images, so the program might continue with degraded performance even if worker images fail progressively.

```
PROGRAM work_share
  USE, INTRINSIC :: iso_fortran_env
  USE :: mod_work, ONLY: work, create_work_item, repeat_work_item, process_item, &
                         WORK_ITEM_EMPTY
  TYPE(event_type) :: submit[*]
  TYPE :: asymmetric_event
     TYPE(event_type), ALLOCATABLE :: event(:)
     LOGICAL, ALLOCATABLE :: available(:)
  END TYPE
  TYPE(asymmetric_event) :: confirm[*]
  TYPE(work) :: work_item[*]
  INTEGER :: count, i, status, work_status

  IF (this_image() == 1) THEN
!
!    set up master-side data structures
     ALLOCATE(confirm%event(2:num_images()))
     ALLOCATE(confirm%available(2:num_images()), SOURCE = .TRUE.)
     DO i = 2, num_images()
        EVENT POST (confirm%event(i))
     END DO
!
!    work distribution loop
     master : DO
        image : DO i = 2, num_images()
           IF (.NOT. confirm%available(i)) CYCLE image
           CALL event_query(confirm%event(i), count, status)
           IF (status == STAT_FAILED_IMAGE) THEN
               confirm%available(i) = .FALSE.
               CYCLE image
           ELSE IF (status /= 0) THEN
               ERROR STOP
           END IF
           IF (count > 0) THEN ! avoid blocking if processing on worker is incomplete
              EVENT WAIT (confirm%event(i), STAT=status)
              IF (status == STAT_FAILED_IMAGE) THEN
```

```
                  confirm%available(i) = .FALSE.
                  CYCLE image
               ELSE IF (status /= 0) THEN
                  ERROR STOP
               END IF

               work_item[i] = create_work_item()

               EVENT POST (submit[i], STAT=status)
               IF (status == STAT_FAILED_IMAGE) THEN
                  CALL repeat_work_item()
                  ! previous item re-created in next iteration
                  confirm%available(i) = .FALSE.
                  CYCLE image
               ELSE IF (status /= 0) THEN
                  ERROR STOP
               END IF
            END IF
         END DO image
         IF ( .NOT. any(confirm%available) ) EXIT master
      END DO master
   ELSE
!
!    work processing loop
      worker : DO
         EVENT WAIT (submit)
         CALL process_item(work_item, work_status)
         IF (work_status == WORK_ITEM_EMPTY) confirm[1]%available(this_image()) = .FALSE.
         ! Notify master that work is done, but check for master having failed
         EVENT POST (confirm[1]%event(this_image()), STAT=status)
         IF (work_status == WORK_ITEM_EMPTY .OR. status /= 0) EXIT worker
      END DO worker
   END IF
END PROGRAM work_share
```

A.2.2 EVENTS example

A tree is a graph in which every node except one has a single "parent" node to which it is connected by an edge. The node without a parent is the "root". The nodes that have a given node as parent are the "children" of that node. The root is at level 1, its children are at level 2, etc.

A multifrontal code to solve a sparse set of linear equations involves a tree. Work at a node starts after work at all its children is complete and their data has been passed to it.

Here we assume that all the nodes have been assigned to images. Each image has a list of its nodes and these are ordered in decreasing tree level (all those at level L preceding those at level $L-1$). For each node, array elements hold the number of children, details about the parent and an event variable. This allows the processing to proceed asynchronously subject to the rule that a parent must wait for all its children as follows:

```
PROGRAM TREE
  USE, INTRINSIC :: ISO_FORTRAN_ENV
  INTEGER,ALLOCATABLE :: NODE(:) ! Tree nodes that this image handles
  INTEGER,ALLOCATABLE :: NC(:)   ! NODE(I) has NC(I) children
  INTEGER,ALLOCATABLE :: PARENT(:), SUB(:)
              ! The parent of NODE(I) is NODE(SUB(I))[PARENT(I)]
  TYPE(EVENT_TYPE),ALLOCATABLE :: DONE(:)[*]
  INTEGER :: I, J, STATUS
! Set up the tree, including allocation of all arrays.
  DO I = 1, SIZE(NODE)
    ! Wait for children to complete
    EVENT WAIT(DONE(I),UNTIL_COUNT=NC(I),STAT=STATUS)
    IF (STATUS/=0) EXIT

    ! Process node, using data from children
    IF (PARENT(I)>0) THEN
       ! Node is not the root.
       ! Place result on image PARENT(I) for node NODE(SUB)[PARENT(I)]
       ! Tell PARENT(I) that this has been done.
       EVENT POST(DONE(SUB(I))[PARENT(I)],STAT=STATUS)
       IF (STATUS/=0) EXIT
    END IF
  END DO
END PROGRAM TREE
```

A.3 Clause 7 notes

A.3.1 Collective subroutine examples

The following example computes a dot product of two scalar coarrays using the co_sum intrinsic to store the result in a noncoarray scalar variable:

```
  subroutine codot(x,y,x_dot_y)
     real :: x[*],y[*],x_dot_y
     x_dot_y = x*y
     call co_sum(x_dot_y)
  end subroutine codot
```

The function below demonstrates passing a noncoarray dummy argument to the co_max intrinsic. The function uses co_max to find the maximum value of the dummy argument across all images. Then the function flags all images that hold values matching the maximum. The function then returns the maximum image index for an image that holds the maximum value:

```
  function find_max(j) result(j_max_location)
    integer, intent(in) :: j
    integer j_max,j_max_location
    call co_max(j,j_max)
! Flag images that hold the maximum j
    if (j==j_max) then
       j_max_location = this_image()
    else
       j_max_location = 0
    end if
! Return highest image index associated with a maximal j
    call co_max(j_max_location)
  end function find_max
```

A.3.2 Atomic memory consistency

A.3.2.1 Relaxed memory model

Parallel programs sometimes have apparently impossible behavior because data transfers and other messages can be delayed, reordered and even repeated, by hardware, communication software, and caching and other forms of optimization. Requiring processors to deliver globally consistent behavior is incompatible with performance on many systems. Fortran specifies that all ordered actions will be consistent (2.3.5 and 8.5 in ISO/IEC 1539-1:2010), but all consistency between unordered segments is deliberately left processor dependent or undefined. Depending on the hardware, this can be observed even when only two images and one mechanism are involved.

A.3.2.2 Examples with atomic operations

When variables are being referenced (atomically) from segments that are unordered with respect to the segment that is is atomically defining or redefining the variables, the results are processor dependent. This supports use of so-called "relaxed memory model" architectures, which can enable more efficient execution on some hardware implementations.

The following examples assume the following declarations:

```
   MODULE example
     USE,INTRINSIC :: ISO_FORTRAN_ENV
     INTEGER(ATOMIC_INT_KIND) :: x[*] = 0, y[*] = 0
```

Example 1:

With `x[j]` and `y[j]` still in their initial state (both zero), image j executes the following sequence of statements:

```
  CALL ATOMIC_DEFINE(x,1)
  CALL ATOMIC_DEFINE(y,1)
```

and image k executes the following sequence of statements:

```
  DO
    CALL ATOMIC_REF(tmp,y[j])
    IF (tmp==1) EXIT
  END DO
  CALL ATOMIC_REF(tmp,x[j])
  PRINT *,tmp
```

The final value of `tmp` on image k can be either 0 or 1. That is, even though image j thinks it wrote `x[j]` before writing `y[j]`, this ordering is not guaranteed on image k.

There are many aspects of hardware and software implementation that can cause this effect, but conceptually this example can be thought of as the change in the value of `y` propagating faster across the inter-image connections than the change in the value of `x`.

Changing the execution on image j by inserting

```
  SYNC MEMORY
```

in between the definitions of `x` and `y` is not sufficient to prevent unexpected results; even though `x` and `y` are being updated in ordered segments, the references from image k are both from a segment that is unordered with respect to image j.

To guarantee the expected value for `tmp` of 1 at the end of the code sequence on image k, it is necessary to ensure that the atomic reference on image k is in a segment that is ordered relative to the segment on image j that defined `x[j]`; `SYNC MEMORY` is certainly necessary, but not sufficient unless it is somehow synchronized.

Example 2:

With the initial state of `x` and `y` on image j (i.e. `x[j]` and `y[j]`) still being zero, execution of

```
    CALL ATOMIC_REF(tmp,x[j])
    CALL ATOMIC_DEFINE(y[j],1)
    PRINT *,tmp
```

on image k1, and execution of

```
    CALL ATOMIC_REF(tmp,y[j])
    CALL ATOMIC_DEFINE(x[j],1)
    PRINT *,tmp
```

on image k2, in unordered segments, might print the value 1 both times.

This can happen by such mechanisms as "load buffering"; one might imagine that what is happening is that the writes (ATOMIC_DEFINE) are overtaking the reads (ATOMIC_REF).

It is likely that insertion of SYNC MEMORY in between the calls to ATOMIC_REF and ATOMIC_DEFINE will be sufficient to prevent this anomalous behavior, but that is only guaranteed by the standard if the SYNC MEMORY executions cause an ordering between the relevant segments on images k1 and k2.

Example 3:

Because there are no segment boundaries implied by collective subroutines, with the initial state as before, execution of

```
    IF (THIS_IMAGE()==1) THEN
      CALL ATOMIC_DEFINE(x[3],23)
      y = 42
    ENDIF
    CALL CO_BROADCAST(y,1)
    IF (THIS_IMAGE()==2) THEN
      CALL ATOMIC_REF(tmp,x[3])
      PRINT *,y,tmp
    END IF
```

could print the values 42 and 0.