TS 18508 Additional Parallel Features in Fortran

J3/13-293

30th June 2013 6:05

Draft document for WG5 Ballot

(Blank page)

Contents

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order to decide whether it will be confirmed for a further three years, revised to become an International Standard, or withdrawn.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again after a further three years, at which time it must either be transformed into an International Standard or be withdrawn.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.

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. 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.

1

2 (Blank page)

3

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*

(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 (7.2)

3.2
team
set of images that access each other's data (5.1).

3.2.1
current team
the team that includes the executing image (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 SUBTEAM statement (5.4).

3.2.4
subteam
a subset of the set of images in a team (5.1).

3.2.5
subteam identifier
integer value identifying a subteam (5.1).

3.2.6
team distance
the distance between a team and one of its ancestors (5.1).

3.3
event variable
scalar variable of type EVENT_TYPE (6.2) from the intrinsic module ISO_FORTRAN_ENV.

3.4
team variable
scalar variable of type TEAM_TYPE (5.2) from 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 access each other's data and synchronize with each other. The current team is the team that includes the executing image. Syntax has been added to *image-selector* (R624 in ISO/IEC 1539-1:2010) to permit specification that image indices are relative to a specified team; otherwise, image indices are relative to the current team. Except by executing a SYNC TEAM statement, synchronization is possible only with other images of the team. Initially, the current team consists of all the images and this is known as the initial team. A team is divided into subteams by executing a FORM SUBTEAM statement. Each subteam is identified by an integer value known as its subteam identifier. Information about the team to which the current image belongs can be determined by the processor from values stored in its team variable.

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.

Within the body of a CHANGE TEAM construct the current team is the subteam specified by the CHANGE TEAM statement.

5.2 TEAM_TYPE

The derived type TEAM_TYPE is an extensible type with no type parameters. Its components are private. A scalar of this type describes a team that includes the executing image. TEAM_TYPE is defined in the intrinsic module ISO_FORTRAN_ENV.

A scalar variable of type TEAM_TYPE is a team variable. A team variable shall not be a coarray or a subcomponent of a coarray.

C501 A team variable shall not appear in a variable definition context except as the *team-variable* in a FORM SUBTEAM statement, as an *allocate-object* in an ALLOCATE statement without a SOURCE=*alloc-opt*, or as an actual argument in a reference to a procedure with an explicit interface where the corresponding dummy argument has INTENT (INOUT).

C502 A variable with a subobject of type TEAM_TYPE shall not have the POINTER attribute and shall not appear in a variable definition context except as an *allocate-object* in an ALLOCATE statement without a SOURCE=*alloc-opt*, or as an actual argument in a reference to a procedure with an explicit interface where the corresponding dummy argument has INTENT (INOUT).

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* ■
■ [, *sync-stat-list*])

R503 *end-change-team-stmt* **is** END TEAM [*team-construct-name*]

R504 *team-variable* **is** *scalar-variable*

C503 (R501) A branch within a CHANGE TEAM construct shall not have a branch target that is outside the construct.

C504 (R501) A RETURN statement shall not appear within a CHANGE TEAM construct.

C505 (R501) A *exit-stmt* or *cycle-stmt* within a CHANGE TEAM construct shall not belong to an outer construct.

C506 (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*.

C507 (R504) A *team-variable* shall be a scalar of the type TEAM_TYPE defined in the ISO_FORTRAN_ENV intrinsic module.

The value of the *team-variable* shall have been formed by executing a FORM SUBTEAM statement. The team executing the *change-team-stmt* shall be the team that formed the team variable value. The current team for the statements of the change-team block is the subteam that was specified for the executing image by the execution of a FORM SUBTEAM statement.

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. When a CHANGE TEAM statement is executed, there is an implicit synchronization of all images of the team identified by *team-variable*; the executing image shall be a member of this team. On each image of the team, execution of the segment following the statement is delayed until all the other 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 images in its team. On each image of the team, execution of the segment following the END TEAM statement is delayed until all the other images of the team have executed the same construct the same number of times.

NOTE 5.1

The deallocation of an allocatable coarray that was not allocated at the beginning of a CHANGE TEAM construct, but was allocated at the end 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 rbracket*

If *team-variable* appears, its value shall be the same as that of a *team-variable* that was assigned a value by a FORM SUBTEAM statement for the current team or an ancestor of the current team, and the cosubscripts are interpreted as if the current team were the team specified by *team-variable*.

5.5 FORM SUBTEAM statement

R505 *form-subteam-stmt* **is** FORM SUBTEAM (*subteam-id*, *team-variable* ■
■ [, *form-subteam-spec-list*])

R506 *subteam-id* **is** *scalar-int-expr*

R507 *form-team-spec* **is** NEW_INDEX = *scalar-int-expr*
or *sync-stat*

C508 (R505) No specifier shall appear more than once in a given *form-subteam-spec-list*.

The FORM SUBTEAM statement defines *team-variable* for a subteam. It is an image control statement. The value of *subteam-id* specifies the subteam to which the executing image belongs. The value of *subteam-id* shall be greater than zero and is the same for all images that are members of the same subteam.

The value of the *scalar-int-expr* in a NEW_INDEX= specifier specifies the image index that the executing image will have in the subteam specified by *subteam-id*. It shall be greater than zero and less than or equal to the number of images in the subteam. Images with the same value for *subteam-id* shall have a different value for the NEW_INDEX= specifier.

If the FORM SUBTEAM statement is executed on one image, it shall be executed by the same statement on all images of the current team, in execution segments that are not ordered with respect to each other. If *team-variable* contains any subscripts, the values of each shall be the same on all these statements.

When a FORM SUBTEAM statement is executed, there is an implicit synchronization of all images in the current team. On these images, execution of the segment following the statement is delayed until all other images in the current team have executed the same statement the same number of times.

The team variable shall not have the value of a team variable for an ancestor of the current team.

NOTE 5.2

Executing the statement

```
FORM SUBTEAM ( 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 subteams according to whether the image index is even or odd.

5.6 SYNC TEAM statement

R508 *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 an execution of FORM SUBTEAM by the current team or an ancestor of the current team. Execution of a SYNC TEAM statement performs a synchronization of the team specified by *team-variable*. Execution on an image, M, of the segment following the SYNC TEAM statement is delayed until each 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 5.3

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 STAT_FAILED_IMAGE

The value of the default integer scalar constant STAT_FAILED_IMAGE is different from the value of STAT_STOPPED_IMAGE, STAT_LOCKED, STAT_LOCKED_OTHER_IMAGE, or STAT_UNLOCKED. 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 variables fail when that variable should be accessible, or the image fails to respond as part of a collective activity. 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.

NOTE 5.4

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.

6 Events

6.1 Introduction

An image can use an EVENT POST statement 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. All components have default initialization. EVENT_TYPE is defined in the ISO_FORTRAN_ENV intrinsic module.

A scalar variable of type EVENT_TYPE is an event variable. An event variable includes a count of the difference between the number of successful posts and successful waits for the event variable. 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 a EVENT POST or EVENT WAIT statement, as an *allocate-object* in an ALLOCATE statement without a SOURCE= *alloc-opt*, 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*, 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 defined in the ISO_FORTRAN_ENV intrinsic module.

A successful post to an event variable increments its count. An unsuccessful post does not change the count.

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* [, *sync-stat-list*])

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

If the count of the *event-variable* is zero, the executing image waits until the count is positive. A successful wait for an event variable decrements its count. Unsuccessful waits do not change the count.

During the execution of the program, the count of an event variable is changed by the execution of a sequence of EVENT POST and EVENT WAIT statements. If the count of an event variable increases through the execution of an EVENT POST statement on image M and later in the sequence decreases through 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 6.1

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 6.2

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, SUBTEAM_ID, and TEAM_DEPTH 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 intrinsics are pure.

The intrinsic procedures THIS_IMAGE and NUM_IMAGES 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 an atomic subroutine is as if the subroutine were executed instantaneously, thus not overlapping other atomic actions that might occur asynchronously. The sequence of atomic actions within ordered segments is specified in 2.3.5 of ISO/IEC 1539-1:2010. How sequences of atomic actions in unordered segments interleave with each other is processor dependent. 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.

7.3 Collective subroutines

A collective subroutine is one that is invoked on each 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 images of the current team in execution segments that are not ordered with respect to each other. From the beginning of execution as the current team, the sequence of invocations of collective subroutines shall be the same on all images of the current team. A call to a collective subroutine shall appear only in a context that allows an image control statement.

If an 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.

All the 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, successful invocation of a collective subroutine causes the argument to become defined with 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 and the effect is otherwise the same as that of executing the SYNC MEMORY statement. 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 had 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.

7.4 New intrinsic procedures

7.4.1 ATOMIC_ADD (ATOM, VALUE [, OLD])

Description. Atomic add operation.

Class. Atomic subroutine.

Arguments.

ATOM shall be scalar and of type integer with kind ATOMIC_INT_KIND, where ATOMIC_INT_KIND is the named constant in the intrinsic module ISO_FORTRAN_ENV. It is an INTENT (INOUT) argument. ATOM becomes defined with the value of ATOM + VALUE.

VALUE shall be scalar and of type integer. It is an INTENT (IN) argument.

OLD (optional) shall be scalar of the same type as ATOM. It is an INTENT (OUT) argument. If it is present, 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 have its 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 [, OLD])

Description. Atomic bitwise AND operation.

Class. Atomic subroutine.

Arguments.

ATOM shall be scalar 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 (optional) shall be scalar of the same type as ATOM. It is an INTENT (OUT) argument. If it is present, 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 scalar 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 the 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 of type logical.

OLD shall be scalar and of the same type 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 prior to the comparison.

7.4.4 ATOMIC_OR (ATOM, VALUE [, OLD])

Description. Atomic bitwise OR operation.

Class. Atomic subroutine.

Arguments.

ATOM shall be scalar 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 (optional) shall be scalar of the same type as ATOM. It is an INTENT (OUT) argument. If it is present, 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 [, OLD])

Description. Atomic bitwise exclusive OR operation.

Class. Atomic subroutine.

Arguments.

ATOM shall be scalar 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 (optional) shall be scalar of the same type as ATOM. It is an INTENT (OUT) argument. If it is present, 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 variable to all images of the current team.

Class. Collective subroutine.

Arguments.

SOURCE shall be a coarray. It is an INTENT(INOUT) argument. 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 of type integer. It is an INTENT(IN) argument. It shall be an image index and have the same value on all images of the current team.

STAT (optional) shall be a scalar 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 is an INTENT(INOUT) argument. 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 and type parameters 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 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 an image index.

STAT (optional) shall be a scalar 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 is an INTENT(INOUT) argument. 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 and type parameters 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 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 an image index.

STAT (optional) shall be a scalar 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 is an INTENT(INOUT) argument. It shall not be polymorphic. If SOURCE is a scalar, the computed value is 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 and type parameters on all images of the current team and each element of the computed value is equal to the value 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 elemental 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 operation. If the operation implemented by OPERATOR is not associative, the computed value of the reduction is processor dependent.

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 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 an image index.

STAT (optional) shall be a scalar 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 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 continues until 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 is an INTENT(INOUT) argument. 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 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 an image index.

STAT (optional) shall be a scalar 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 [, STATUS])

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 default integer. It is an INTENT(OUT) argument. If the invocation is successful, COUNT becomes defined with the difference between the number of successful posts and successful waits for EVENT. Otherwise, it is given the value 0.

STATUS (optional) shall be scalar and of type default integer. It is an INTENT(OUT) argument. It becomes defined with value 0 if the invocation is successful and with a processor-defined nonzero value if the invocation is unsuccessful.

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 and 2 successful waits to EVENT[2], after the invocation

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

COUNT has the value 8.

NOTE 7.1

Execution of EVENT_QUERY does not imply any synchronization.

7.4.12 FAILED_IMAGES ([KIND])

Description. Indices of failed images.

Class. Transformational function.

Argument. 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.

Result Value. The elements of the result are the values of the image indices of the failed images in the current 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 SUBTEAM_ID ([DISTANCE])

Description. Subteam 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 subteam 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 subteam 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 SUBTEAM_ID to control which code is executed.

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

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) :: SUBTEAM

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

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

7.5 Modified intrinsic procedures

7.5.1 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 result value is the number of images 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 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 LOGICAL argument. Its value determines whether the result is the number of failed images or the number of nonfailed images. If DISTANCE is present, the result applies to the team it specifies, otherwise the result applies to the current team. If FAILED is present with the value true, the result is the number of failed images in the applicable team, otherwise the result is the total number of nonfailed images in the applicable team.

7.5.2 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 in the team of which the invoking image was last 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.

(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 to restrict the image set of remote memory references, coarray allocations, and synchronizations to a subset of all the images of the program;

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

• atomic memory operations provide powerful low-level primitives for synchronization of activities among images;

• 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.

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)

1.3.145a
team
set of images that access each others data (2.3.4).

1.3.145a.1
current team
the team that includes the executing image (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 SUBTEAM statement (8.5.2c).

1.3.145a.4
subteam
a subset of the set of images in a team (2.3.4).

1.3.145a.5
subteam identifier
integer value identifying a subteam (2.3.4).

1.3.145a.6
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

{At the end of 2.3.4 Program execution insert three new paragraphs}

A team of images is a set of images that access each other's data and synchronize with each other. The current team is the team that includes the executing image. Unless *team-variable* is specified in an *image-selector* (R624), all image indices are relative to the current team. Except by executing a SYNC TEAM statement (8.5.5a), synchronization is possible only with other images of the team. Initially, the current team consists of all the images and this is known as the initial team. A team is divided into subteams by executing a FORM SUBTEAM statement (8.5.2c). Each subteam is identified by an integer value known as its subteam identifier. Information about the team to which the current image belongs can be determined by the processor from values stored in its team variable.

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.

Within the body of a CHANGE TEAM construct (8.1.4a) the current team is the subteam specified by the CHANGE TEAM statement.

8.5 Edits to clause 6

{In 6.6 Image selectors, replace R624 with}

R624 *image-selector* **is** *lbracket* [*team-variable* ::] *cosubscript-list rbracket*

{In 6.6 Image selectors, after paragraph 2 insert}

If *team-variable* appears, its value shall be the same as that of a *team-variable* that was assigned a value by a FORM SUBTEAM (8.5.2c) statement for the current team or an ancestor of the current team, and the cosubscripts are interpreted as if the current team were the team specified by *team-variable*.

{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 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 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 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 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 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.

8.6 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.}

{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 SUBTEAM;

• SYNC TEAM;

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

All image control statements except CRITICAL, END CRITICAL, FORM SUBTEAM, 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.}

{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 SUBTEAM 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 images in the current team. Execution on an image, M, of the segment following the SYNC ALL statement is delayed until each other 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}

The appearance of a STAT= or ERRMSG= specifier in a CHANGE TEAM statement is treated as an appearance both there and in the corresponding END TEAM statement.

If the STAT= specifier appears, successful execution of a CHANGE TEAM, END TEAM, FORM SUBTEAM, LOCK, SYNC ALL, SYNC IMAGES, SYNC MEMORY, or UNLOCK statement causes the specified variable to become defined with the value zero.

If the STAT= specifier appears in a CHANGE TEAM, END TEAM, FORM SUBTEAM, LOCK, SYNC ALL, SYNC IMAGES, SYNC MEMORY, or UNLOCK statement and its execution is not successful, the specified variable becomes defined with a nonzero value and the effect is otherwise the same as that of executing the SYNC MEMORY statement. If there is a stopped image in the current team, the variable becomes defined with the constant STAT_STOPPED_IMAGE in the intrinsic module ISO_FORTRAN_ENV (13.8.2); otherwise, if no image of the current team has been detected as stopped or failed, the variable becomes defined with 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 an image had been detected as failed, the variable becomes defined with the the constant STAT_FAILED_IMAGE.

{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, FORM SUBTEAM, LOCK, SYNC ALL, SYNC IMAGES, SYNC MEMORY, or UNLOCK statement and its execution is not successful, error termination is initiated.

If an ERRMSG= specifier appears in a CHANGE TEAM, END TEAM, FORM SUBTEAM, LOCK, SYNC ALL, SYNC IMAGES, SYNC MEMORY, 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.

8.7 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.

{In 13.1 Classes of intrinsic procedures, insert six new paragraphs following paragraph 3 and Note 13.1}

A collective subroutine is one that is invoked on each 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 images of the current team in execution segments that are not ordered with respect to each other. From the beginning of execution as the current team, the sequence of calls to collective subroutines shall be the same on all images of the current team. A call to a collective subroutine shall appear only in a context that allows an image control statement.

If an 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.

All the collective subroutines have the optional arguments STAT and ERRMSG.

If the STAT argument is present, successful invocation of a collective subroutine causes the argument to become defined with 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 and the effect is otherwise the same as that of executing the SYNC MEMORY statement. If execution involves synchronization with an image that has stopped, the argument becomes defined with the value of STAT_STOPPED_IMAGE in the intrinsic module ISO_FORTRAN_ENV (13.8.2); 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 (13.8.2). If an image had been detected as failed, but no other error condition occurred, the argument is assigned the value of the constant STAT_FAILED_IMAGE.

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.

{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 [,OLD])	A Atomic ADD operation.

ATOMIC_AND (ATOM, VALUE [,OLD])	A	Atomic bitwise AND operation.
ATOMIC_CAS (ATOM, OLD, COMPARE, NEW)	A	Atomic compare and swap.
ATOMIC_OR (ATOM, VALUE [,OLD])	A	Atomic bitwise OR operation.
ATOMIC_XOR (ATOM, VALUE [,OLD])	A	Atomic bitwise exclusive OR operation.
CO_BROADCAST (SOURCE, SOURCE_IMAGE)	C	Copy a variable to all images.
CO_MAX (SOURCE [, RESULT, RESULT_IMAGE])	C	Compute maximum of elements on all images.
CO_MIN (SOURCE [, RESULT, RESULT_IMAGE])	C	Compute minimum of elements on all images.
CO_REDUCE (SOURCE, OPERATOR [, RESULT, RESULT_IMAGE])	C	General reduction of elements on all images.
CO_SUM (SOURCE [, RESULT, RESULT_IMAGE])	C	Sum elements on all images.
EVENT_QUERY (EVENT, COUNT[, STATUS])	S	Count of an event.
FAILED_IMAGES ([KIND])	T	Indices of failed images.
SUBTEAM_ID ([DISTANCE])	T	Subteam identifier.
TEAM_DEPTH ( )	T	Team depth for this image.

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

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 7.3.1 through 7.3.14 of this Technical Specification in order alphabetically, with subcaluse numbers adjusted accordingly.}

{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 LOGICAL argument. 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 result value is the number of images 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 team specified is the team of which invoking image was a member with a team distance of DISTANCE from the current team; otherwise, the team specified is the initial team.

If DISTANCE is present, the result applies to the team it specifies, otherwise the result applies to the current team. If FAILED is present with the value true, the result is the number of failed images in the applicable team,

otherwise the result is the total number of nonfailed images in the applicable team.

{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 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 13.8.2 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.}

{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.}

{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.}

8.8 Edits to clause 16

{At the end of the list of variable definition contexts in 16.6.7p1, replace the "." at the end of entry (15) with ";" and add two new entries as follows}

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

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

8.9 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 computed value of the CO_SUM intrinsic function;
- the computed value of the CO_REDUCE intrinsic function.

Annex A

(Informative)

Extended notes

A.1 Clause 5 notes

Example: 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)    :: SUBTEAM_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 SUBTEAM(MY_SURFACE_TYPE, SUBTEAM_SURFACE_TYPE)

   CHANGE TEAM(SUBTEAM_SURFACE_TYPE)
      SELECT CASE (SUBTEAM_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.2 Clause 6 notes

Example 1: Use of EVENT_QUERY.

```
USE,INTRINSIC :: ISO_FORTRAN_ENV
INTEGER               :: COUNT, STATUS
```

```
TYPE(EVENT_TYPE      ) :: EVENT[*]

CALL EVENT_QUERY(EVENT, COUNT, STATUS)
IF (STATUS /= 0) THEN
   PRINT*,'PROBLEM WITH EVENT QUERYING'
ELSE
   IF (COUNT == 0) THEN
      ! Do some useful work not related to the event.
   ELSE
      EVENT WAIT(EVENT, STAT=STATUS)
      IF (STATUS /= 0) THEN
         PRINT*,'PROBLEM WITH EVENT WAITING'
      ELSE
         ! Do the work related to the event.
      ENDIF
   ENDIF
ENDIF
```

Example 2: Producer consumer program.

```
PROGRAM PROD_CONS
USE, INTRINSIC :: ISO_FORTRAN_ENV
INTEGER :: I, COUNT, STATUS
TYPE(EVENT_TYPE) :: EVENT[*]
DO
   DO I = 1, NUM_IMAGES()
      CALL EVENT_QUERY(EVENT[I], COUNT, STATUS)
      IF (STATUS /= 0) THEN
         PRINT*,'PROBLEM QUERYING EVENT'
      ELSE
         IF (I /= THIS_IMAGE()) THEN
            IF (COUNT == 0) THEN
               ! Produce some work
               EVENT POST(EVENT[I], STATUS)
               IF (STATUS /= 0) THEN
                  PRINT*,'PROBLEM POSTING EVENT'
               ENDIF
            ENDIF
         ELSE
            EVENT WAIT(EVENT, STATUS)
            IF (STATUS /= 0) THEN
               PRINT*,'PROBLEM WAITING FOR EVENT'
            ELSE
               ! Consume some work
            ENDIF
         ENDIF
      ENDIF
   ENDDO
ENDDO
END PROD_CONS
```

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
```