

# TS 18508 Additional Parallel Features in Fortran

WG5/N1983

3rd July 2013 13:45

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 . . . . .	10
5.5	FORM SUBTEAM statement . . . . .	10
5.6	SYNC TEAM statement . . . . .	11
5.7	STAT_FAILED_IMAGE . . . . .	11
6	Events . . . . .	13
6.1	Introduction . . . . .	13
6.2	EVENT_TYPE . . . . .	13
6.3	EVENT POST statement . . . . .	13
6.4	EVENT WAIT statement . . . . .	13
7	Intrinsic procedures . . . . .	15
7.1	General . . . . .	15
7.2	Atomic subroutines . . . . .	15
7.3	Collective subroutines . . . . .	15
7.4	New intrinsic procedures . . . . .	16
7.4.1	ATOMIC_ADD (ATOM, VALUE [, OLD]) . . . . .	16
7.4.2	ATOMIC_AND (ATOM, VALUE [, OLD]) . . . . .	16
7.4.3	ATOMIC_CAS (ATOM, OLD, COMPARE, NEW) . . . . .	16
7.4.4	ATOMIC_OR (ATOM, VALUE [, OLD]) . . . . .	17
7.4.5	ATOMIC_XOR (ATOM, VALUE [, OLD]) . . . . .	17
7.4.6	CO_BROADCAST (SOURCE, SOURCE_IMAGE [, STAT, ERRMSG]) . . . . .	17
7.4.7	CO_MAX (SOURCE [, RESULT, RESULT_IMAGE, STAT, ERRMSG]) . . . . .	18
7.4.8	CO_MIN (SOURCE [, RESULT, RESULT_IMAGE, STAT, ERRMSG]) . . . . .	18
7.4.9	CO_REDUCE (SOURCE, OPERATOR [, RESULT, RESULT_IMAGE, STAT, ERRMSG]) . . . . .	19
7.4.10	CO_SUM (SOURCE [, RESULT, RESULT_IMAGE, STAT, ERRMSG]) . . . . .	20
7.4.11	EVENT_QUERY ( EVENT, COUNT [, STATUS] ) . . . . .	21
7.4.12	FAILED_IMAGES ([KIND]) . . . . .	21
7.4.13	SUBTEAM_ID ([DISTANCE]) . . . . .	21
7.4.14	TEAM_DEPTH( ) . . . . .	22
7.5	Modified intrinsic procedures . . . . .	23
7.5.1	NUM_IMAGES . . . . .	23
7.5.2	THIS_IMAGE . . . . .	23

8	Required editorial changes to ISO/IEC 1539-1:2010(E)	25
8.1	General	25
8.2	Edits to Introduction	25
8.3	Edits to clause 1	25
8.4	Edits to clause 2	26
8.5	Edits to clause 6	26
8.6	Edits to clause 8	27
8.7	Edits to clause 13	29
8.8	Edits to clause 16	31
8.9	Edits to annex A	31
Annex A	(informative) Extended notes	33
A.1	Clause 5 notes	33
A.2	Clause 6 notes	33
A.3	Clause 7 notes	35
A.3.1	Collective subroutine examples	35

## 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 facilities 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

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

1

2

(Blank page)

3

2



## 1 2 Normative references

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

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

1

2

(Blank page)

3

4

## 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.

1

2

(Blank page)

3

**6**

## 1 **4 Compatibility**

### 2 **4.1 New intrinsic procedures**

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

### 7 **4.2 Fortran 2008 compatibility**

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

1

2

(Blank page)

3

**8**

## 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. 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 ]
```

- 1 R504 *team-variable* is *scalar-variable*
- 2 C503 (R501) A branch within a CHANGE TEAM construct shall not have a branch target that is outside the  
3 construct.
- 4 C504 (R501) A RETURN statement shall not appear within a CHANGE TEAM construct.
- 5 C505 (R501) A *exit-stmt* or *cycle-stmt* within a CHANGE TEAM construct shall not belong to an outer  
6 construct.
- 7 C506 (R501) If the *change-team-stmt* of a *change-team-construct* specifies a *team-construct-name*, the corres-  
8 ponding *end-change-team-stmt* shall specify the same *team-construct-name*. If the *change-team-stmt* of a  
9 *change-team-construct* does not specify a *team-construct-name*, the corresponding *end-change-team-stmt*  
10 shall not specify a *team-construct-name*.
- 11 C507 (R504) A *team-variable* shall be a scalar of the type TEAM\_TYPE defined in the ISO\_FORTRAN\_ENV  
12 intrinsic module.

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

17 An allocatable coarray that was allocated when execution of a *change-team* construct began shall not be deal-  
18 located during the execution of the construct. An allocatable coarray that is allocated when execution of a  
19 *change-team* construct completes is deallocated if it was not allocated when execution of the construct began.

20 The CHANGE TEAM and END TEAM statements are image control statements. When a CHANGE TEAM  
21 statement is executed, there is an implicit synchronization of all images of the team identified by *team-variable*;  
22 the executing image shall be a member of this team. On each image of the team, execution of the segment following  
23 the statement is delayed until all the other images of the team have executed the same statement the same number  
24 of times. When a CHANGE TEAM construct completes execution, there is an implicit synchronization of all  
25 images in its team. On each image of the team, execution of the segment following the END TEAM statement  
26 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.

## 27 5.4 Image selectors

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

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

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

## 33 5.5 FORM SUBTEAM statement

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

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



1 R507 *form-team-spec* is NEW\_INDEX = *scalar-int-expr*  
 2 or *sync-stat*

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

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

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

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

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

17 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.

## 18 5.6 SYNC TEAM statement

19 R508 *sync-team-stmt* is SYNC TEAM ( *team-variable* [, *sync-stat-list*] )

20 The SYNC TEAM statement is an image control statement. The value of *team-variable* shall have been established  
 21 by an execution of FORM SUBTEAM by the current team or an ancestor of the current team. Execution of  
 22 a SYNC TEAM statement performs a synchronization of the team specified by *team-variable*. Execution on an  
 23 image, M, of the segment following the SYNC TEAM statement is delayed until each other image of the specified  
 24 team has executed a SYNC TEAM statement specifying the same team as many times as has image M. The  
 25 segments that executed before the SYNC TEAM statement on an image precede the segments that execute after  
 26 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.

## 27 5.7 STAT\_FAILED\_IMAGE

28 The value of the default integer scalar constant STAT\_FAILED\_IMAGE is different from the value of STAT\_  
 29 STOPPED\_IMAGE, STAT\_LOCKED, STAT\_LOCKED\_OTHER\_IMAGE, or STAT\_UNLOCKED. If the pro-  
 30 cessor has the ability to detect that an image of the current team has failed and does so, the value of STAT\_  
 31 FAILED\_IMAGE is assigned to the variable specified in a STAT=specifier in an execution of an image control  
 32 statement, or the STAT argument in an invocation of a collective procedure. A failed image is one for which

1 references or definitions of variables fail when that variable should be accessible, or the image fails to respond as  
2 part of a collective activity. A failed image remains failed for the remainder of the program execution. If more  
3 than one nonzero status value is valid for the execution of a statement, the status variable is defined with a value  
4 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.

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

3 During the execution of the program, the count of an event variable is changed by the execution of a sequence of  
4 EVENT POST and EVENT WAIT statements. If the count of an event variable increases through the execution  
5 of an EVENT POST statement on image M and later in the sequence decreases through the execution of an  
6 EVENT WAIT statement on image T, the segments preceding the EVENT POST statement on image M precede  
7 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`.

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

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

## 6 **7.4 New intrinsic procedures**

### 7 **7.4.1 ATOMIC\_ADD (ATOM, VALUE [, OLD])**

8 **Description.** Atomic add operation.

9 **Class.** Atomic subroutine.

#### 10 **Arguments.**

11 ATOM shall be scalar and of type integer with kind ATOMIC\_INT\_KIND, where ATOMIC\_INT\_KIND  
12 is the named constant in the intrinsic module ISO\_FORTRAN\_ENV. It is an INTENT (INOUT)  
13 argument. ATOM becomes defined with the value of ATOM + VALUE.

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

15 OLD (optional) shall be scalar of the same type as ATOM. It is an INTENT (OUT) argument. If it is present,  
16 it is defined with the value of ATOM that was used for performing the ADD operation.

#### 17 **Examples.**

18 CALL ATOMIC\_ADD(I[3], 42) causes the value of I on image 3 to have its to become its previous value plus 42.

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

### 21 **7.4.2 ATOMIC\_AND (ATOM, VALUE [, OLD])**

22 **Description.** Atomic bitwise AND operation.

23 **Class.** Atomic subroutine.

#### 24 **Arguments.**

25 ATOM shall be scalar and of type integer with kind ATOMIC\_INT\_KIND, where ATOMIC\_INT\_KIND is a  
26 named constant in the intrinsic module ISO\_FORTRAN\_ENV. It is an INTENT (INOUT) argument.  
27 ATOM becomes defined with the value IAND(ATOM,INT(VALUE,ATOMIC\_INT\_KIND)).

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

29 OLD (optional) shall be scalar of the same type as ATOM. It is an INTENT (OUT) argument. If it is present,  
30 it is defined with the value of ATOM that was used for performing the bitwise AND operation.

31 **Example.** CALL ATOMIC\_AND (I[3], 6, Iold) causes I on image 3 to become defined with the value 4 and the  
32 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  
33 the bitwise AND operation executed.

### 34 **7.4.3 ATOMIC\_CAS (ATOM, OLD, COMPARE, NEW)**

35 **Description.** Atomic compare and swap.

36 **Class.** Atomic subroutine.

#### 37 **Arguments.**

1     **ATOM**       shall be scalar and of type integer with kind `ATOMIC_INT_KIND` or of type logical with kind  
2     `ATOMIC_LOGICAL_KIND`, where `ATOMIC_INT_KIND` and `ATOMIC_LOGICAL_KIND` are the  
3     named constants in the intrinsic module `ISO_FORTRAN_ENV`. It is an `INTENT (INOUT)` argu-  
4     ment. If the value of `ATOM` is equal to the value of `COMPARE`, `ATOM` becomes defined with the  
5     value of `INT (NEW, ATOMIC_INT_KIND)` if it is of type integer, and with the value of `NEW` if it  
6     of type logical.

7     **OLD**        shall be scalar and of the same type as `ATOM`. It is an `INTENT (OUT)` argument. It is defined  
8     with the value of `ATOM` that was used for performing the compare operation.

9     **COMPARE** shall be scalar and of the same type and kind as `ATOM`. It is an `INTENT(IN)` argument.

10    **NEW**        shall be scalar and of the same type as `ATOM`. It is an `INTENT(IN)` argument.

11    **Example.** `CALL ATOMIC_CAS(I[3], OLD, Z, 1)` causes `I` on image 3 to become defined with the value 1 if its  
12    value is that of `Z`, and `OLD` to be defined with the value of `I` on image 3 prior to the comparison.

#### 13    **7.4.4    **ATOMIC\_OR (ATOM, VALUE [, OLD])****

14    **Description.** Atomic bitwise OR operation.

15    **Class.** Atomic subroutine.

##### 16    **Arguments.**

17    **ATOM**       shall be scalar and of type integer with kind `ATOMIC_INT_KIND`, where `ATOMIC_INT_KIND` is a  
18    named constant in the intrinsic module `ISO_FORTRAN_ENV`. It is an `INTENT (INOUT)` argument.  
19    `ATOM` becomes defined with the value `IOR(ATOM,INT(VALUE,ATOMIC_INT_KIND))`.

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

21    **OLD** (optional) shall be scalar of the same type as `ATOM`. It is an `INTENT (OUT)` argument. If it is present,  
22    it is defined with the value of `ATOM` that was used for performing the bitwise OR operation.

23    **Example.** `CALL ATOMIC_OR (I[3], 1, Iold)` causes `I` on image 3 to become defined with the value 3 and the  
24    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  
25    the bitwise OR operation executed.

#### 26    **7.4.5    **ATOMIC\_XOR (ATOM, VALUE [, OLD])****

27    **Description.** Atomic bitwise exclusive OR operation.

28    **Class.** Atomic subroutine.

##### 29    **Arguments.**

30    **ATOM**       shall be scalar and of type integer with kind `ATOMIC_INT_KIND`, where `ATOMIC_INT_KIND` is a  
31    named constant in the intrinsic module `ISO_FORTRAN_ENV`. It is an `INTENT (INOUT)` argument.  
32    `ATOM` becomes defined with the value `IEOR(ATOM,INT(VALUE,ATOMIC_INT_KIND))`.

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

34    **OLD** (optional) shall be scalar of the same type as `ATOM`. It is an `INTENT (OUT)` argument. If it is present, it  
35    is defined with the value of `ATOM` that was used for performing the bitwise exclusive OR operation.

36    **Example.** `CALL ATOMIC_XOR (I[3], 1, Iold)` causes `I` on image 3 to become defined with the value 2 and the  
37    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  
38    the bitwise exclusive OR operation executed.

#### 39    **7.4.6    **CO\_BROADCAST (SOURCE, SOURCE\_IMAGE [, STAT, ERRMSG])****

40    **Description.** Copy a variable to all images of the current team.

41    **Class.** Collective subroutine.

1 **Arguments.**

2 SOURCE shall be a coarray. It is an INTENT(INOUT) argument. SOURCE becomes defined, as if by intrinsic  
3 assignment, on all images of the current team with the value of SOURCE on image SOURCE\_  
4 IMAGE.

5 SOURCE\_IMAGE shall be of type integer. It is an INTENT(IN) argument. It shall be an image index and have  
6 the same value on all images of the current team.

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

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

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

10 **Example.** If SOURCE is the array [1, 5, 3] on image one, after execution of CALL CO\_BROADCAST(SOURCE,1)  
11 the value of SOURCE on all images of the current team is [1, 5, 3].

12 **7.4.7 CO\_MAX (SOURCE [, RESULT, RESULT\_IMAGE, STAT, ERRMSG])**

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

14 **Class.** Collective subroutine.

15 **Arguments.**

16 SOURCE shall be of type integer, real, or character. It is an INTENT(INOUT) argument. If it is a scalar,  
17 the computed value is equal to the maximum value of SOURCE on all images of the current team.  
18 If it is an array it shall have the same shape and type parameters on all images of the current team  
19 and each element of the computed value is equal to the maximum value of all the corresponding  
20 elements of SOURCE on the images of the current team.

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

23 RESULT\_IMAGE (optional) shall be of type integer. It is an INTENT(IN) argument. If it is present, it shall be  
24 present on all images of the current team, have the same value on all images of the current team,  
25 and that value shall be an image index.

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

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

28 If RESULT and RESULT\_IMAGE are not present, the computed value is assigned to SOURCE on all the images  
29 of the current team. If RESULT is not present and RESULT\_IMAGE is present, the computed value is assigned to  
30 SOURCE on image RESULT\_IMAGE and SOURCE on all other images of the current team becomes undefined.  
31 If RESULT is present and RESULT\_IMAGE is not present, the computed value is assigned to RESULT on all  
32 images of the current team. If RESULT and RESULT\_IMAGE are present, the computed value is assigned to  
33 RESULT on image RESULT\_IMAGE and RESULT on all other images of the current team becomes undefined.  
34 If RESULT is present, SOURCE is not modified.

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

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

39 **7.4.8 CO\_MIN (SOURCE [, RESULT, RESULT\_IMAGE, STAT, ERRMSG])**

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

41 **Class.** Collective subroutine.

42 **Arguments.**



1 SOURCE shall be of type integer, real, or character. It is an INTENT(INOUT) argument. If it is a scalar,  
 2 the computed value is equal to the minimum value of SOURCE on all images of the current team.  
 3 If it is an array it shall have the same shape and type parameters on all images of the current team  
 4 and each element of the computed value is equal to the minimum value of all the corresponding  
 5 elements of SOURCE on the images of the current team.

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

8 RESULT\_IMAGE (optional) shall be of type integer. It is an INTENT(IN) argument. If it is present, it shall be  
 9 present on all images of the current team, have the same value on all images of the current team,  
 10 and that value shall be an image index.

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

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

13 If RESULT and RESULT\_IMAGE are not present, the computed value is assigned to SOURCE on all the images  
 14 of the current team. If RESULT is not present and RESULT\_IMAGE is present, the computed value is assigned to  
 15 SOURCE on image RESULT\_IMAGE and SOURCE on all other images of the current team becomes undefined.  
 16 If RESULT is present and RESULT\_IMAGE is not present, the computed value is assigned to RESULT on all  
 17 images of the current team. If RESULT and RESULT\_IMAGE are present, the computed value is assigned to  
 18 RESULT on image RESULT\_IMAGE and RESULT on all other images of the current team becomes undefined.  
 19 If RESULT is present, SOURCE is not modified.

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

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

## 24 7.4.9 CO\_REDUCE (SOURCE, OPERATOR [, RESULT, RESULT\_IMAGE, STAT, ER- 25 RMSG])

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

27 **Class.** Collective subroutine.

### 28 Arguments.

29 SOURCE is an INTENT(INOUT) argument. It shall not be polymorphic. If SOURCE is a scalar, the  
 30 computed value is the reduction operation of applying OPERATOR to the values of SOURCE on  
 31 all images of the current team. If SOURCE is an array it shall have the same shape and type  
 32 parameters on all images of the current team and each element of the computed value is equal to  
 33 the value of the reduction operation of applying OPERATOR to all the corresponding elements of  
 34 SOURCE on all the images of the current team.

35 OPERATOR shall be a pure elemental function with two arguments of the same type and type parameters as  
 36 SOURCE. Its result shall have the same type and type parameters as SOURCE. The arguments  
 37 and result shall not be polymorphic. OPERATOR shall implement a mathematically commutative  
 38 operation. If the operation implemented by OPERATOR is not associative, the computed value of  
 39 the reduction is processor dependent.

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

42 RESULT\_IMAGE (optional) shall be of type integer. It is an INTENT(IN) argument. If it is present, it shall be  
 43 present on all images of the current team, have the same value on all images of the current team,  
 44 and that value shall be an image index.

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

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

1 If RESULT and RESULT\_IMAGE are not present, the computed value is assigned to SOURCE on all the images  
2 of the current team. If RESULT is not present and RESULT\_IMAGE is present, the computed value is assigned to  
3 SOURCE on image RESULT\_IMAGE and SOURCE on all other images of the current team becomes undefined.  
4 If RESULT is present and RESULT\_IMAGE is not present, the computed value is assigned to RESULT on all  
5 images of the current team. If RESULT and RESULT\_IMAGE are present, the computed value is assigned to  
6 RESULT on image RESULT\_IMAGE and RESULT on all other images of the current team becomes undefined.  
7 If RESULT is present, SOURCE is not modified.

8 The computed value of a reduction operation over a set of values is the result of an iterative process. Each  
9 iteration involves the execution of  $r = \text{OPERATOR}(x, y)$  for  $x$  and  $y$  in the set, the removal of  $x$  and  $y$  from the  
10 set, and the addition of  $r$  to the set. The process continues until the set has only one element which is the value  
11 of the reduction.

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

13 **Example.** If the number of images in the current team is two and SOURCE is the array [1, 5, 3] on one image  
14 and [4, 1, 6] on the other image, and MyADD is a function that returns the sum of its two integer arguments,  
15 the value of RESULT after executing the statement CALL CO\_REDUCE(SOURCE, MyADD, RESULT) is [5,  
16 6, 9] on both images.

#### 17 7.4.10 CO\_SUM (SOURCE [, RESULT, RESULT\_IMAGE, STAT, ERRMSG])

18 **Description.** Sum elements on the current team of images.

19 **Class.** Collective subroutine.

##### 20 **Arguments.**

21 SOURCE shall be of numeric type. It is an INTENT(INOUT) argument. If it is a scalar, the computed value  
22 is equal to a processor-dependent and image-dependent approximation to the sum of the values of  
23 SOURCE on all images of the current team. If it is an array it shall have the same shape on all  
24 images of the current team and each element of the computed value is equal to a processor-dependent  
25 and image-dependent approximation to the sum of all the corresponding elements of SOURCE on  
26 the images of the current team.

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

29 RESULT\_IMAGE (optional) shall be of type integer. It is an INTENT(IN) argument. If it is present, it shall be  
30 present on all images of the current team, have the same value on all images of the current team,  
31 and that value shall be an image index.

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

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

34 If RESULT and RESULT\_IMAGE are not present, the computed value is assigned to SOURCE on all the images  
35 of the current team. If RESULT is not present and RESULT\_IMAGE is present, the computed value is assigned to  
36 SOURCE on image RESULT\_IMAGE and SOURCE on all other images of the current team becomes undefined.  
37 If RESULT is present and RESULT\_IMAGE is not present, the computed value is assigned to RESULT on all  
38 images of the current team. If RESULT and RESULT\_IMAGE are present, the computed value is assigned to  
39 RESULT on image RESULT\_IMAGE and RESULT on all other images of the current team becomes undefined.  
40 If RESULT is present, SOURCE is not modified.

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

42 **Example.** If the number of images in the current team is two and SOURCE is the array [1, 5, 3] on one image  
43 and [4, 1, 6] on the other image, the value of RESULT after executing the statement CALL CO\_SUM(SOURCE,  
44 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.

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

2 **Result Characteristics.** Default integer scalar.

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

8 **Example.** The following code illustrates the use of SUBTEAM\_ID to control which code is executed.

```

9 TYPE(Team_Type) :: ODD_EVEN
10 :
11 ME = THIS_IMAGE()
12 FORM SUBTEAM ( 2-MOD(ME,2), ODD_EVEN )
13 CHANGE TEAM (ODD_EVEN)
14   SELECT CASE (SUBTEAM_ID())
15     CASE (1)
16       : ! Code for odd images in parent team
17     CASE (2)
18       : ! Code for even images in parent team
19   END SELECT
20 END TEAM

```

#### 21 7.4.14 TEAM\_DEPTH( )

22 **Description.** Team depth for the current team.

23 **Class.** Transformational function.

24 **Arguments.** None.

25 **Result Characteristics.** Scalar default integer.

26 **Result Value.** The result of TEAM\_DEPTH is an integer with a value equal to the team distance between the  
27 current team and the initial team.

28 **Example.**

```

29 PROGRAM TD
30   USE, INTRINSIC :: ISO_FORTRAN_ENV
31   INTEGER          :: I_TEAM_DEPTH
32   TYPE(Team_Type) :: SUBTEAM
33
34   FORM SUBTEAMS(1, SUBTEAM)
35   CHANGE TEAM(SUBTEAM)
36     I_TEAM_DEPTH = TEAM_DEPTH()
37   END TEAM
38 END

```

39 On completion of the CHANGE TEAM construct, I\_TEAM\_DEPTH has the value 1.

## 1 **7.5 Modified intrinsic procedures**

### 2 **7.5.1 NUM\_IMAGES**

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

5 The DISTANCE argument shall be a nonnegative scalar integer. If DISTANCE is not present the result value is  
6 the number of images in the current team.

7 If DISTANCE is present with a value less than or equal to the team distance between the current team and the  
8 initial team, the team specified is the team of which the invoking image was a member with a team distance of  
9 DISTANCE from the current team; otherwise, the team specified is the initial team.

10 The FAILED argument shall be a scalar LOGICAL argument. Its value determines whether the result is the  
11 number of failed images or the number of nonfailed images. If DISTANCE is present, the result applies to the  
12 team it specifies, otherwise the result applies to the current team. If FAILED is present with the value true, the  
13 result is the number of failed images in the applicable team, otherwise the result is the total number of nonfailed  
14 images in the applicable team.

### 15 **7.5.2 THIS\_IMAGE**

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

18 The DISTANCE argument shall be a scalar integer. It shall be nonnegative. If DISTANCE is not present, the  
19 result value is the image index of the invoking image in the current team. If DISTANCE is present with a value  
20 less than or equal to the team distance between the current team and the initial team, the result has the value of  
21 the image index in the team of which the invoking image was last a member with a team distance of DISTANCE  
22 from the current team; otherwise, the result has the value of the image index that the invoking image had in the  
23 initial team.

1

2

(Blank page)

3

## 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 **1.3.145a.1**  
 2 **current team**  
 3 the team that includes the executing image (2.3.4).
- 4 **1.3.145a.2**  
 5 **initial team**  
 6 the current team when the program began execution (2.3.4).
- 7 **1.3.145a.3**  
 8 **parent team**  
 9 team from which the current team was formed by executing a FORM SUBTEAM statement (8.5.2c).
- 10 **1.3.145a.4**  
 11 **subteam**  
 12 a subset of the set of images in a team (2.3.4).
- 13 **1.3.145a.5**  
 14 **subteam identifier**  
 15 integer value identifying a subteam (2.3.4).
- 16 **1.3.145a.6**  
 17 **team distance**  
 18 the distance between a team and one of its ancestors (2.3.4).
- 19 **1.3.154.1-**  
 20 **event variable**  
 21 scalar variable of type EVENT\_TYPE (13.8.2.8a) from the intrinsic module ISO\_FORTRAN\_ENV.
- 22 **1.3.154.3**  
 23 **team variable**  
 24 scalar variable of type TEAM\_TYPE (13.8.2.26) from the intrinsic module ISO\_FORTRAN\_ENV.

## 25 8.4 Edits to clause 2

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

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

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

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

## 39 8.5 Edits to clause 6

40 {In 6.6 Image selectors, replace R624 with}

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

---



1 {In 6.6 Image selectors, after paragraph 2 insert}

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

---

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

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

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

---

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

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

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

## 23 8.6 Edits to clause 8

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

- 25 • CHANGE TEAM construct;
- 

26 {Following 8.1.4 BLOCK construct insert 5.3 CHANGE TEAM construct from this Technical Specification as  
27 8.1.4a, with rule, constraint, and Note numbers modified.}

---

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

- 30 • CHANGE TEAM and END TEAM;
  - 31 • EVENT POST and EVENT WAIT;
  - 32 • FORM SUBTEAM;
  - 33 • SYNC TEAM;
- 

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

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

---

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

38 A coarray that is of type EVENT\_TYPE may be referenced or defined during the execution of a segment that is

1 unordered relative to the execution of another segment in which that coarray of type `EVENT_TYPE` is defined.

---

2 {Following 8.5.2 Segments insert 6.3 `EVENT POST` statement from this Technical Specification as 8.5.2a, with  
3 rule and constraint numbers modified.}

---

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

---

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

---

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

9 Execution of a `SYNC ALL` statement performs a synchronization of all images in the current team. Execution  
10 on an image, *M*, of the segment following the `SYNC ALL` statement is delayed until each other image in the  
11 team has executed a `SYNC ALL` statement as many times as has image *M*. The segments that executed before  
12 the `SYNC ALL` statement on an image precede the segments that execute after the `SYNC ALL` statement on  
13 another image.

---

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

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

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

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

---

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

---

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

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

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

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

---

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

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

41 If an `ERRMSG=` specifier appears in a `CHANGE TEAM`, `END TEAM`, `FORM SUBTEAM`, `LOCK`, `SYNC ALL`,

1 SYNC IMAGES, SYNC MEMORY, or UNLOCK statement and its execution is not successful, the processor  
 2 shall assign an explanatory message to the specified variable. If the execution is successful, the processor shall  
 3 not change the value of the variable.

## 4 8.7 Edits to clause 13

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

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

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

10 For invocation of an atomic subroutine with an argument OLD, the assignment of the value to OLD is not part  
 11 of the atomic action. For invocation of an atomic subroutine, evaluation of an INTENT(IN) argument is not part  
 12 of the atomic action.

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

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

20 If an argument to a collective subroutine is a whole coarray the corresponding ultimate arguments on all images  
 21 of the current team shall be corresponding coarrays as described in 2.4.7.

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

23 If the STAT argument is present, successful invocation of a collective subroutine causes the argument to become  
 24 defined with the value zero.

25 If the STAT argument is present in an invocation of a collective subroutine and an error condition occurs, the  
 26 argument is assigned a nonzero value and the effect is otherwise the same as that of executing the SYNC MEMORY  
 27 statement. If execution involves synchronization with an image that has stopped, the argument becomes defined  
 28 with the value of STAT\_STOPPED\_IMAGE in the intrinsic module ISO\_FORTRAN\_ENV (13.8.2); otherwise, if  
 29 no image of the current team has stopped or failed, the argument is assigned a processor-dependent positive value  
 30 that is different from the value of STAT\_STOPPED\_IMAGE or STAT\_FAILED\_IMAGE in the intrinsic module  
 31 ISO\_FORTRAN\_ENV (13.8.2). If an image had been detected as failed, but no other error condition occurred,  
 32 the argument is assigned the value of the constant STAT\_FAILED\_IMAGE.

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

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

38 C indicates that the procedure is a collective subroutine

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

40 ATOMIC\_ADD (ATOM, VALUE [,OLD])                    A Atomic ADD operation.

41 ATOMIC\_AND (ATOM, VALUE [,OLD])                    A Atomic bitwise AND operation.

1	ATOMIC_CAS (ATOM, OLD, COMPARE, NEW)	A	Atomic compare and swap.
2	ATOMIC_OR (ATOM, VALUE [,OLD])	A	Atomic bitwise OR operation.
3	ATOMIC_XOR (ATOM, VALUE [,OLD])	A	Atomic bitwise exclusive OR operation.
4	CO_BROADCAST (SOURCE, SOURCE_IMAGE)	C	Copy a variable to all images.
5	CO_MAX (SOURCE [, RESULT, RESULT_IMAGE])	C	Compute maximum of elements on all images.
6	CO_MIN (SOURCE [, RESULT, RESULT_IMAGE])	C	Compute minimum of elements on all images.
7	CO_REDUCE (SOURCE, OPERATOR [, RESULT, RESULT_IMAGE])	C	General reduction of elements on all images.
8			
9	CO_SUM (SOURCE [, RESULT, RESULT_IMAGE])	C	Sum elements on all images.
10	EVENT_QUERY (EVENT, COUNT[, STATUS])	S	Count of an event.
11	FAILED_IMAGES ([KIND])	T	Indices of failed images.
12	SUBTEAM_ID ([DISTANCE])	T	Subteam identifier.
13	TEAM_DEPTH ( )	T	Team depth for this image.
<hr/>			
14	{In 13.5 Standard generic intrinsic procedures, Table 13.1, edit the entries for NUM_IMAGES() and THIS_IMAGE() as follows}		
15			
16	NUM_IMAGES	([ <u>DISTANCE</u> , <u>FAILED</u> ])	T Number of images.
17	THIS_IMAGE	([ <u>DISTANCE</u> ])	T Index of the invoking image.
<hr/>			
18	{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.}		
19			
<hr/>			
20	{In 13.7.126 NUM_IMAGES, edit the subclause title as follows}		
21	13.7.126 NUM_IMAGES ([ <u>DISTANCE</u> , <u>FAILED</u> ])		
<hr/>			
22	{In 13.7.126 NUM_IMAGES, replace paragraph 3 with}		
23	<b>Arguments.</b>		
24	DISTANCE (optional) shall be a nonnegative scalar integer. It is an INTENT(IN) argument.		
25	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.		
26			
<hr/>			
27	{In 13.7.126 NUM_IMAGES, replace paragraph 5 with}		
28	<b>Result Value.</b> If DISTANCE is not present the result value is the number of images in the current team.		
29	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.		
30			
31			
32	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.		
33			
34			
<hr/>			

1 {In 13.7.165 THIS\_IMAGE ( ) or THIS\_IMAGE (COARRAY [, DIM]) edit the subclause title as follows }

2 13.7.165 THIS IMAGE ([DISTANCE]) or THIS IMAGE (COARRAY [, DIM])

---

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

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

---

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

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

---

13 {In 13.8.2 The ISO\_FORTRAN\_ENV intrinsic module, insert a new subclause 13.8.2.8a consisting of subclause  
14 6.2 EVENT\_TYPE of this Technical Specification, but omitting the final sentence of the first paragraph. }

---

15 {In 13.8.2 The ISO\_FORTRAN\_ENV intrinsic module, insert a new subclause 13.8.2.21b consisting of subclause  
16 5.6 STAT\_FAILED\_IMAGE of this Technical Specification. }

---

17 {In 13.8.2 The ISO\_FORTRAN\_ENV intrinsic module, append a new subclause 13.8.2.26 consisting of subclause  
18 5.2 TEAM\_TYPE of this Technical Specification, but omitting the final sentence of the first paragraph. }

## 19 8.8 Edits to clause 16

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

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

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

## 24 8.9 Edits to annex A

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

- 27 • the conditions that cause an image to fail;
- 28 • the computed value of the CO\_SUM intrinsic function;
- 29 • 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.

```

7  SUBROUTINE COMPUTE_FLUXES(FLUX_MOM, FLUX_SENS, FLUX_LAT)
8  USE, INTRINSIC :: ISO_FORTRAN_ENV
9  REAL, INTENT(OUT) :: FLUX_MOM(:, :), FLUX_SENS(:, :), FLUX_LAT(:, :)
10 INTEGER, PARAMETER :: LAND=1, SEA=2, ICE=3
11 CHARACTER(LEN=10) :: SURFACE_TYPE
12 INTEGER           :: MY_SURFACE_TYPE, N_IMAGE
13 TYPE(Team_Type)  :: SubTeam_Surface_Type
14
15     CALL GET_SURFACE_TYPE(THIS_IMAGE(), SURFACE_TYPE) ! Surface type
16     SELECT CASE (SURFACE_TYPE)                       ! of the executing image
17     CASE ('LAND')
18         MY_SURFACE_TYPE = LAND
19     CASE ('SEA')
20         MY_SURFACE_TYPE = SEA
21     CASE ('ICE')
22         MY_SURFACE_TYPE = ICE
23     CASE DEFAULT
24         ERROR STOP
25     END SELECT
26     FORM SubTeam(My_Surface_Type, SubTeam_Surface_Type)
27
28     CHANGE Team(SubTeam_Surface_Type)
29     SELECT CASE (SubTeam_ID( ))
30     CASE (LAND)    ! Compute fluxes over land surface
31         CALL COMPUTE_FLUXES_LAND(FLUX_MOM, FLUX_SENS, FLUX_LAT)
32     CASE (SEA)    ! Compute fluxes over sea surface
33         CALL COMPUTE_FLUXES_SEA(FLUX_MOM, FLUX_SENS, FLUX_LAT)
34     CASE (ICE)    ! Compute fluxes over ice surface
35         CALL COMPUTE_FLUXES_ICE(FLUX_MOM, FLUX_SENS, FLUX_LAT)
36     CASE DEFAULT
37         ERROR STOP
38     END SELECT
39     END Team
40 END SUBROUTINE COMPUTE_FLUXES

```

### A.2 Clause 6 notes

Example 1: Use of EVENT\_QUERY.

```

43 USE, INTRINSIC :: ISO_FORTRAN_ENV
44 INTEGER           :: COUNT, STATUS

```

```

1  TYPE(EVENT_TYPE      ) :: EVENT[*]
2
3  CALL EVENT_QUERY(EVENT, COUNT, STATUS)
4  IF (STATUS /= 0) THEN
5      PRINT*, 'PROBLEM WITH EVENT QUERYING'
6  ELSE
7      IF (COUNT == 0) THEN
8          ! Do some useful work not related to the event.
9      ELSE
10         EVENT WAIT(EVENT, STAT=STATUS)
11         IF (STATUS /= 0) THEN
12             PRINT*, 'PROBLEM WITH EVENT WAITING'
13         ELSE
14             ! Do the work related to the event.
15         ENDIF
16     ENDIF
17 ENDIF

```

18 Example 2: Producer consumer program.

```

19 PROGRAM PROD_CONS
20 USE, INTRINSIC :: ISO_FORTRAN_ENV
21 INTEGER :: I, COUNT, STATUS
22 TYPE(EVENT_TYPE) :: EVENT[*]
23 DO
24     DO I = 1, NUM_IMAGES()
25         CALL EVENT_QUERY(EVENT[I], COUNT, STATUS)
26         IF (STATUS /= 0) THEN
27             PRINT*, 'PROBLEM QUERYING EVENT'
28         ELSE
29             IF (I /= THIS_IMAGE()) THEN
30                 IF (COUNT == 0) THEN
31                     ! Produce some work
32                     EVENT POST(EVENT[I], STATUS)
33                     IF (STATUS /= 0) THEN
34                         PRINT*, 'PROBLEM POSTING EVENT'
35                     ENDIF
36                 ENDIF
37             ELSE
38                 EVENT WAIT(EVENT, STATUS)
39                 IF (STATUS /= 0) THEN
40                     PRINT*, 'PROBLEM WAITING FOR EVENT'
41                 ELSE
42                     ! Consume some work
43                 ENDIF
44             ENDIF
45         ENDIF
46     ENDDO
47 ENDDO
48 END PROD_CONS

```



## 1 A.3 Clause 7 notes

### 2 A.3.1 Collective subroutine examples

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

```
5     subroutine codot(x,y,x_dot_y)
6         real :: x[*],y[*],x_dot_y
7         x_dot_y = x*y
8         call co_sum(x_dot_y)
9     end subroutine codot
```

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

```
14     function find_max(j) result(j_max_location)
15         integer, intent(in) :: j
16         integer j_max,j_max_location
17         call co_max(j,j_max)
18     ! Flag images that hold the maximum j
19         if (j==j_max) then
20             j_max_location = this_image()
21         else
22             j_max_location = 0
23         end if
24     ! Return highest image index associated with a maximal j
25         call co_max(j_max_location)
26     end function find_max
```