# TS 18508 Additional Parallel Features in Fortran

J3/12-201

**19th October 2012 16:14**

This is an internal working document of J3.

(Blank page)

# Contents

# List of Tables

# 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order to decide whether it will be confirmed for a further three years, revised to become an International Standard, or withdrawn.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again after a further three years, at which time it must either be transformed into an International Standard or be withdrawn.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 29113:2012 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 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 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 | Clause 5 |
| Events | Clause 6 |
| New 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 |

(Blank page)

# 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.

(Blank page)

# 2 Normative references

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

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

(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.

**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)

(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 Partitions and teams of images

## 5.1 Image partitions

Description of partitioning images.

## 5.2 Image teams

Description of teams of images.

(Blank page)

# 6 Events

## 6.1 Event variables

Description of event variables.

## 6.2 Event usage

Description of facilities to use events.

(Blank page)

# 7 New intrinsic procedures

## 7.1 General

Detailed specifications of the generic intrinsic subroutines ATOMIC_ADD, ATOMIC_AND, ATOMIC_CAS, ATOMIC_OR, ATOMIC XOR, CO_BROADCAST, CO_MAX, CO_MIN, CO_REDUCE, and CO_SUM are provided in 7.3. The types and type parameters of the arguments to these intrinsic subroutines are determined by these specifications. The "Argument" paragraphs specify requirements on the actual arguments of the procedures. All of these intrinsics are pure.

## 7.2 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 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 of ISO/IEC 1539-1:2010.

## 7.3 New intrinsic subroutines

### 7.3.1 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 becomes 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 becomes defined with 99 if the previous value of M was 99 on image 4.

### 7.3.2 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 becomes 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 become defined with the value 5 if the value of I[3] was 5 when the bitwise AND operation executed.

### 7.3.3 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 becomes 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 become defined with the value of I on image 3 prior to the comparison.

### 7.3.4 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 becomes defined with the value of ATOM that was used for performing the bitwise OR operation.

**Example.** CALL ATOMIC_XOR (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 become defined with the value 2 if the value of I[3] was 2 when the bitwise OR operation executed.

### 7.3.5 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 becomes 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 become defined with the value 3 if the value of I[3] was 3 when the bitwise exclusive XOR operation executed.

### 7.3.6 CO_BROADCAST (SOURCE, SOURCE_IMAGE)

**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.

**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].

### 7.3.7 CO_MAX (SOURCE [, RESULT, RESULT_IMAGE])

**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.

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.

**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.

### 7.3.8 CO_MIN (SOURCE [, RESULT, RESULT_IMAGE])

**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.

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.

**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.

### 7.3.9 CO_REDUCE (SOURCE, OPERATOR [, RESULT, RESULT_IMAGE])

**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.

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.

**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.

### 7.3.10 CO_SUM (SOURCE [, RESULT, RESULT_IMAGE])

**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.

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.

**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.

(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.

## 8.3 Edits to clause 13

{In 13.1 Classes of intrinsic procedures}

Move the two paragraphs of subclause 7.2 in this Technical Specificationto follow paragraph 3 and Note 13.1 in subclause 13.1 of ISO/IEC 1539-1:2010.

---

{In 13.5 Standard generic intrinsic procedures, para 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. |

---

{Move subclauses 7.3.1 through 7.3.10 in this Technical Specification to subclause 13.7 of ISO/IEC 1539-1:2010 in order alphabetically}

## 8.4 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 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 7 notes

### A.1.1 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:

```
  real, save :: x[*],y[*],xy[*]
  real x_dot_y
!Initialize x and y
  x = this_image()
  call random_number(y)
  xy = x*y
  call co_sum(xy,x_dot_y)
```

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
    j_max_location = merge(this_image(),0,j==j_max)
! Return highest image index associated with a maximal j
    call co_max(j_max_location)
  end function find_max
```