OpenMP openmp.org

OpenMP 5.2 API Syntax Reference Guide

The OpenMP® API is a scalable model that gives parallel programmers a simple and flexible interface for developing portable parallel applications in C/C++ and Fortran.

OpenMP is suitable for a wide range of algorithms running on multicore nodes and chips, NUMA systems, GPUs, and other such devices attached to a CPU.

C/C++ C/C++ content | *Fortran* or *For* Fortran content | [n.n.n] Sections in 5.2. spec | [n.n.n] Sections in 5.1. spec | 9 See Clause info on pg. 9

Getting Started

Navigating this reference guide

Directives and Constructs 1
Clauses 9
Runtime Library Routines 10
Environment Variables 14
ICV values 15
Using OpenMP Tools 16

OpenMP Examples Document

An Examples Document and a link to a GitHub repository with code samples is at link.openmp.org/examples51.

OpenMP directive syntax

A directive is a combination of the base-language mechanism and a *directive-specification* (the *directive-name* followed by optional clauses). A construct consists of a directive and, often, additional base language code.

C/C++ C directives are formed exclusively with pragmas. C++ directives are formed from either pragmas or attributes.

Fortran Fortran directives are formed with comments in free form and fixed form sources (codes).

Examples:

C/C++ **#pragma omp** *directive-specification*

C++ **[[omp :: directive(** *directive-specification* **)]]**

C++ **[[using omp : directive(** *directive-specification* **)]]**

For **!$omp** *directive-specification*

For **!$omp** *directive-specification*
!$omp end *directive-name*

Directives and Constructs

OpenMP constructs consist of a directive and, if defined in the syntax, an associated structured block that follows. • OpenMP directives except **simd** and any declarative directive may not appear in Fortran PURE procedures. • *structured-block* is a construct or block of executable statements with a single entry at the top and a single exit at the bottom. • *strictly-structured-block* is a structured block that is a Fortran BLOCK construct. • *loosely-structured-block* is a structured block that isn't strictly structured and doesn't start with a Fortran BLOCK construct. • *omp-integer-expression* is of a *C/C++* scalar int type or *Fortran* scalar integer type. • *omp-logical-expression* is a *C/C++* scalar expression or *Fortran* logical expression.

Data environment directives

threadprivate [5.2] [2.21.2]

Specifies that variables are replicated, with each thread having its own copy. Each copy of a **threadprivate** variable is initialized once prior to the first reference to that copy.

C/C++	**#pragma omp threadprivate (***list***)**
Fortran	**!$omp threadprivate (***list***)**

list:

C/C++ A comma-separated list of file-scope, namespace-scope, or static block-scope variables that do not have incomplete types.

For A comma-separated list of named variables and named common blocks. Common block names must appear between slashes.

declare reduction [5.5.11] [2.21.5.7]

Declares a *reduction-identifier* that can be used in a **reduction**, **in_reduction**, or **task_reduction** clause.

C/C++	**#pragma omp declare reduction (** *reduction-identifier* **:** *typename-list* **:** *combiner***) ** *[initializer-clause]*
Fortran	**!$omp declare reduction &** **(***reduction-identifier* **:** *type-list* **:** *combiner***)** *[initializer-clause]*

combiner:

C/C++ An expression

For An assignment statement or a subroutine name followed by an argument list.

initializer-clause: **initializer (***initializer-expr***)**

initializer-expr: **omp_priv** = *initializer* or *function-name* **(***argument-list***)**

reduction-identifier:

C/C++ A base language identifier (for *C*), or an *id-expression* (for *C++*), or one of the following operators: **+**, *****, **&**, **|**, **^**, **&&**, **||**

For A base language identifier, user-defined operator, or one of the following operators: **+**, *****, **.and.**, **.or.**, **.eqv.**, **.negv.**; or one of the following intrinsic procedure names: **max**, **min**, **iand**, **ior**, **ieor**.

C/C++ typename-list: A list of type names

For type-list: A list of type specifiers that must not be **CLASS(*)** or abstract type.

scan [5.6] [2.11.8]

Specifies that scan computations update the list items on each iteration of an enclosing loop nest associated with a worksharing-loop, worksharing-loop SIMD, or **simd** directive.

C/C++:
```
{
   structured-block-sequence
   #pragma omp scan clause
   structured-block-sequence
}
```

Fortran:
```
structured-block-sequence
!$omp scan  clause
structured-block-sequence
```

clause:
exclusive (*list***)**
inclusive (*list***)**

declare mapper [5.8.8] [2.21.7.4]

Declares a user-defined mapper for a given type, and may define a *mapper-identifier* for use in a **map** clause.

C/C++	**#pragma omp declare mapper (***[mapper-identifier* **:** *]* **** *type var***)** *[clause[[,] clause] ...]*
Fortran	**!$omp declare mapper (***[mapper-identifier***:** *]type* **::** *var***) &** *[clause[[,] clause] ...]*

clause:
map (*[[map-modifier*, *[map-modifier*, ... *]] map-type* **:** *] list***)** 9

map-type: **alloc**, **from**, **to**, **tofrom**

map-modifier: **always**, **close**, **present**, **mapper(default)**, **iterator(***iterator-definitions***)**

mapper-identifier:
A base-language identifier or **default** 9

type: A valid type in scope

var: A valid base-language identifier

Memory management directives

Memory spaces [6.1] [2.13.1]

Predefined memory spaces represent storage resources for storage and retrieval of variables.

Memory space	Storage selection intent
omp_default_mem_space	Default storage
omp_large_cap_mem_space	Large capacity
omp_const_mem_space	Variables with constant values
omp_high_bw_mem_space	High bandwidth
omp_low_lat_mem_space	Low latency

allocate [6.6] [2.13.3]

Specifies how a set of variables is allocated.

C/C++	**#pragma omp allocate (***list***)** *[clause[[*,*]clause] ...]*
Fortran	**!$omp allocate (***list***)** *[clause[[*,*]clause] ...]*

clause:

align (*alignment***)**
alignment: An integer power of 2.

allocator (*allocator***)**
allocator:
C/C++ type **omp_allocator_handle_t**
For *kind* **omp_allocator_handle_kind**

allocators [6.7]

Specifies that OpenMP memory allocators are used for certain variables that are allocated by the associated *allocate-stmt*.

Fortran:
```
!$omp allocators [clause[ [,]clause] ... ]
    allocate-stmt
[ !$omp end allocators ]
```

clause: **allocate** 9

allocate-stmt: A Fortran ALLOCATE statement.

Variant directives

[begin]metadirective [7.4.3, 7.4.4] [2.3.4]

A directive that can specify multiple directive variants, one of which may be conditionally selected to replace the **metadirective** based on the enclosing OpenMP context.

C/C++:
```
#pragma omp metadirective [clause[ [,] clause] ... ]
- or -
#pragma omp begin metadirective [clause[ [,] clause] ... ]
    stmt(s)
#pragma omp end metadirective
```

Fortran:
```
!$omp metadirective [clause[ [,] clause] ... ]
- or -
!$omp begin metadirective [clause[ [,] clause] ... ]
    stmt(s)
!$omp end metadirective
```

clause:

when (*context-selector-specification*: *[directive-variant]***)**
Conditionally select a directive variant.

otherwise (*[directive-variant]***)**
Conditionally select a directive variant. **otherwise** was named **default** in previous versions.

Continued

Directives and Constructs (continued)

[begin]declare variant [7.5.4-5] [2.3.5]

Declares a specialized variant of a base function and the context in which it is used.

C/C++:
```
#pragma omp declare variant(variant-func-id) \
    clause [[ [,] clause] ... ]
[#pragma omp declare variant(variant-func-id) \
    clause [[ [,] clause] ... ]
[ ... ]]
    function definition or declaration
- or -
#pragma omp begin declare variant clause-match
    declaration-definition-seq
#pragma omp end declare variant
```

Fortran:
```
!$omp declare variant ([base-proc-name : ] &
    variant-proc-name) clause  [[ [,] clause] ... ]
```

clause:

adjust_args (*adjust-op* **:** *argument-list*)

adjust-op: **nothing**, **need_device_ptr**

append_args (*append-op*[[, *append-op*]...])

append-op: **interop** (*interop-type* [[, *interop-type*]...])

enter

link

match (*context-selector-specification*)

REQUIRED. Specifies how to adjust the arguments of the base function when a specified variant function is selected for replacement.

C/C++ *variant-func-id*

The name of a function variant that is a base language identifier, or for C++, a *template-id*.

For *variant-proc-name*

The name of a function variant that is a base language identifier.

clause-match:

match (*context-selector-specification*)

REQUIRED **match** clause

dispatch [7.6] [2.3.6]

Controls whether variant substitution occurs for a function call in the structured block.

C/C++:
```
#pragma omp dispatch [clause [ [,] clause] ... ]
    function-dispatch-structured-block
```

Fortran:
```
!$omp dispatch [clause [ [,] clause] ... ]
    function-dispatch-structured-block
[!$omp end dispatch]
```

clause:

depend ([*depend-modifier*,] *dependence-type* **:** *locator-list*) 9

device (*omp-integer-expression*) 9

Identifies the target device that is associated with a device construct.

is_device_ptr (*list*)

list: device pointers

nocontext (*omp-logical-expression*)

If *omp-logical-expression* evaluates to true, the construct is not added to the construct set of the OpenMP context.

novariants (*omp-logical-expression*)

If *omp-logical-expression* evaluates to true, no function variant is selected for the call in the applicable **dispatch** region.

nowait 9

declare simd [7.7] [2.11.5.3]

Applied to a function or subroutine to enable creation of one or more versions to process multiple arguments using SIMD instructions from a single invocation in a SIMD loop.

C/C++:
```
#pragma omp declare simd [clause[ [,]clause] ... ]
[#pragma omp declare simd [clause[ [, clause] ... ] ]
[... ]
    function definition or declaration
```

Fortran:
```
!$omp declare simd [(proc-name)]  [clause[ [,]clause] ... ]
```

clause:

aligned (*argument-list*[**:** *alignment*])

Declares one or more list items to be aligned to the specified number of bytes.

alignment: Optional constant positive integer expression

inbranch

linear (*linear-list*[**:** *linear-step*]) 9

notinbranch

simdlen (*length*)

Specifies the preferred number of iterations to be executed concurrently.

uniform (*argument-list*)

Declares arguments to have an invariant value for all concurrent invocations of the function in the execution of a single SIMD loop.

[begin]declare target [7.8.1-2] [2.14.7]

A declarative directive that specifies that variables, functions, and subroutines are mapped to a device.

C/C++:
```
#pragma omp declare target (extended-list)
- or -
#pragma omp declare target clause[ [,]clause ... ]
- or -
#pragma omp begin declare target \
        [clause[[,]clause] ... ]
    declarations-definition-seq
#pragma omp end declare target
```

Fortran:
```
!$omp declare target (extended-list)
- or -
!$omp declare target [clause[ [,]clause] ...]
```

clause:

device_type (**host** | **nohost** | **any**)

enter (*extended-list*)

A comma-separated list of named variables, procedure names, and named common blocks.

indirect[(*invoked-by-fptr*)]

Determines if the procedures in an **enter** clause may be invoked indirectly.

link (*list*)

Supports compilation of functions called in a **target** region that refer to the *list* items.

- For the second *C/C++* form of **declare target**, at least one clause must be **enter** or **link**.
- For **begin declare target**, the **enter** and **link** clauses are not permitted.

Informational and utility directives

requires [8.2] [2.5.1]

Specifies the features that an implementation must provide in order for the code to compile and to execute correctly.

C/C++:
```
#pragma omp requires clause [ [ [,] clause] ... ]
```

Fortran:
```
!$omp requires clause [ [ [,] clause] ... ]
```

clause:

atomic_default_mem_order (**seq_cst** | **acq_rel** | **relaxed**)

dynamic_allocators

Enables memory allocators to be used in a **target** region without specifying the **uses_allocators** clause on the corresponding **target** construct. (See **target** on page 5 of this guide.)

reverse_offload

Requires an implementation to guarantee that if a **target** construct specifies a **device** clause in which the **ancestor** modifier appears, the **target** region can execute on the parent device of an enclosing **target** region. (See **target** on page 5.)

unified_address

Requires that all devices accessible through OpenMP API routines and directives use a unified address space.

unified_shared_memory

Guarantees that in addition to the requirement of **unified_address**, storage locations in memory are accessible to threads on all available devices.

assume, [begin]assumes [8.3.2-4] [2.5.2]

Provides invariants to the implementation that may be used for optimization purposes.

C/C++:
```
#pragma omp assumes clause [ [ [,] clause] ... ]
- or -
#pragma omp begin assumes clause [ [ [,] clause] ... ]
    declaration-definition-seq
#pragma omp end assumes
- or -
#pragma omp assume clause [ [ [,] clause] ... ]
    structured-block
```

Fortran:
```
!$omp assumes clause [ [ [,] clause] ... ]
- or -
!$omp assume clause [ [ [,] clause] ... ]
    loosely-structured-block
!$omp end assume
- or -
!$omp assume clause [ [ [,] clause] ... ]
    strictly-structured-block
[ !$omp end assume]
```

clause:

absent (*directive-name* [[, *directive-name*] ...])

Lists directives absent in the scope.

contains (*directive-name* [[, *directive-name*] ...])

Lists directives likely to be in the scope.

holds (*omp-logical-expression*)

An *expression* guaranteed to be true in the scope.

no_openmp

Indicates that no OpenMP code is in the scope.

no_openmp_routines

Indicates that no OpenMP runtime library calls are executed in the scope.

no_parallelism

Indicates that no OpenMP tasks or SIMD constructs will be executed in the scope.

Continued

Directives and Constructs (continued)

nothing [8.4] [2.5.3]

Indicates explicitly that the intent is to have no effect.

C/C++	**#pragma omp nothing**
Fortran	**!$omp nothing**

error [8.5] [2.5.4]

Instructs the compiler or runtime to display a message and to perform an error action.

C/C++	**#pragma omp error** *[clause [[,] clause] ...]*
Fortran	**!$omp error** *[clause [[,] clause] ...]*

clause:

- **at (compilation | execution)**
- **message (***msg-string***)**
- **severity (fatal | warning)**

Loop transformation constructs

tile [9.1] [2.11.9.1]

Tiles one or more loops.

C/C++	**#pragma omp tile** *clause* *loop-nest*
Fortran	**!$omp tile** *clause* *loop-nest* *[***!$omp end tile***]*

clause: **sizes (***size-list***)**

unroll [9.2] [2.11.9.2]

Fully or partially unrolls a loop.

C/C++	**#pragma omp unroll** *[clause]* *loop-nest*
Fortran	**!$omp unroll** *[clause]* *loop-nest* *[***!$omp end unroll***]*

clause:

- **full**
- **partial** *[***(***unroll-factor***)***]*

Parallelism constructs

parallel [10.1] [2.6]

Creates a team of OpenMP threads that execute the region.

C/C++	**#pragma omp parallel** *[clause[[,]clause] ...]* *structured-block*
Fortran	**!$omp parallel** *[clause[[,]clause] ...]* *loosely-structured-block* **!$omp end parallel** *- or -* **!$omp parallel** *[clause[[,]clause] ...]* *strictly-structured-block* *[* **!$omp end parallel***]*

clause:

- **allocate** 9
- **copyin (***list***)**
- **default (***data-sharing-attribute***)** 9
- **firstprivate (***list***)** 9
- **num_threads (***nthreads***)**
  - Specifies the number of threads to execute.
- **private (***list***)** 9
- **proc_bind (close | primary | spread)**
  - **close**: Instructs the execution environment to assign the threads in the team to places close to the place of the parent thread.
  - **primary**: Instructs the execution environment to assign every thread in the team to the same place as the primary thread.
  - **spread**: Creates a sparse distribution for a team of *T* threads among the *P* places of the parent's place partition.
- **reduction** 9
- **shared (***list***)** 9

teams [10.2] [2.7]

Creates a league of initial teams where the initial thread of each team executes the region.

C/C++	**#pragma omp teams** *[clause[[,]clause] ...]* *structured-block*
Fortran	**!$omp teams** *[clause[[,]clause] ...]* *loosely-structured-block* **!$omp end teams** *- or -* **!$omp teams** *[clause[[,]clause] ...]* *strictly-structured-block* *[***!$omp end teams***]*

clause:

- **allocate** 9
- **default (***data-sharing-attribute***)** 9
- **firstprivate (***list***)** 9
- **num_teams (***[lower-bound* **:** *] upper-bound***)**
- **private (***list***)** 9
- **reduction** 9
- **shared (***list***)** 9
- **thread_limit (***omp-integer-expression***)**

simd [10.4] [2.11.5.1]

Applied to a loop to indicate that the loop can be transformed into a SIMD loop.

C/C++	**#pragma omp simd** *[clause[[,]clause] ...]* *loop-nest*
Fortran	**!$omp simd** *[clause[[,]clause] ...]* *loop-nest* *[***!$omp end simd***]*

clause:

- **aligned (***list[* **:** *alignment]***)**
  - Declares one or more list items to be aligned to the specified number of bytes.
  - *alignment*: Optional constant positive integer expression
- **collapse (***n***)** 9
- **if (***[***simd :** *] omp-logical-expression***)** 9
- **lastprivate (***[lastprivate-modifier* **:** *] list***)** 9
- **linear (***list[* **:** *linear-step]***)** 9
- **nontemporal (***list***)**
  - Accesses to the storage locations in *list* have low temporal locality across the iterations in which those storage locations are accessed.
- **order (***[order-modifier* **:** *]* **concurrent)** 9
  - *order-modifier*: **reproducible** or **unconstrained**
- **private (***list***)** 9
- **reduction** 9
- **safelen (***length***)**
  - If used then no two iterations executed concurrently with SIMD instructions can have a greater distance in the logical iteration space than the value of *length*.
- **simdlen (***length***)**
  - Specifies the preferred number of iterations to be executed concurrently.

masked [10.5] [2.8]

Specifies a structured block that is executed by a subset of the threads of the current team.

C/C++	**#pragma omp masked** *[clause]* *structured-block*
Fortran	**$omp masked** *[clause]* *loosely-structured-block* **!$omp end masked** *- or -* **!$omp masked** *[clause]* *strictly-structured-block* *[* **!$omp end masked***]*

clause:

- **filter (***thread_num***)**
  - Selects which thread executes.

Work-distribution constructs

single [11.1] [2.10.2]

Specifies that the associated structured block is executed by only one of the threads in the team.

C/C++	**#pragma omp single** *[clause[[,]clause] ...]* *structured-block*
Fortran	**!$omp single** *[clause[[,]clause] ...]* *loosely-structured-block* **!$omp end single** *[end-clause[[,]end-clause] ...]* *- or -* **!$omp single** *[clause[[,]clause] ...]* *strictly-structured-block* *[***!$omp end single** *[end-clause[[,]end-clause] ...]]*

clause:

- **allocate** 9
- **copyprivate (***list***)**
- **firstprivate (***list***)** 9
- **nowait** 9
- **private (***list***)** 9

end-clause:

- **copyprivate (***list***)**
- **nowait** 9

workshare [11.4] [2.10.3]

Divides the execution of the enclosed structured block into separate units of work, each executed only once by one thread.

Fortran	**!$omp workshare** *[clause]* *loosely structured-block* **!$omp end workshare** *[clause]* *- or -* **!$omp workshare** *[clause]* *strictly structured-block* *[***!$omp end workshare** *[clause]]*

clause:

- **nowait** 9

scope [11.2] [2.9]

Defines a structured block that is executed by all threads in a team but where additional OpenMP operations can be specified.

C/C++	**#pragma omp scope** *[clause[[,]clause] ...]* *structured-block*
Fortran	**!$omp scope** *[clause[[,]clause] ...]* *loosely-structured-block* **!$omp end scope** *[***nowait***]* *- or -* **!$omp scope** *[clause[[,]clause] ...]* *strictly-structured-block* *[***!$omp end scope** *[***nowait***]]*

clause:

- **allocate** 9
- **firstprivate (***list***)** 9
- **nowait** 9
- **private (***list***)** 9
- **reduction** 9

Continued

Directives and Constructs (continued)

section and sections [11.3] [2.10.1]

A non-iterative worksharing construct that contains a set of structured blocks that are to be distributed among and executed by the threads in a team.

C/C++:
```
#pragma omp sections [clause[ [,] clause] ... ]
  {
  [#pragma omp section]
      structured-block-sequence
  [#pragma omp section
      structured-block-sequence]
  ...
  }
```

Fortran:
```
!$omp sections [clause[ [,] clause] ... ]
  [!$omp section]
      structured-block-sequence
  [!$omp section
      structured-block-sequence]
  ...
!$omp end sections [nowait]
```

clause:

- **allocate** 9
- **firstprivate** (*list*) 9
- **lastprivate** ([*lastprivate-modifier*:] *list*) 9
- **nowait** 9
- **private** (*list*) 9
- **reduction** 9

do and for [11.5.1-2] [2.11.4]

Specifies that the iterations of associated loops will be executed in parallel by threads in the team.

C/C++:
```
#pragma omp for [clause[ [,]clause] ... ]
  loop-nest
```

Fortran:
```
!$omp do [clause[ [,]clause] ... ]
  loop-nest
[!$omp end do [nowait] ]
```

clause:

- **allocate** 9
- **collapse** (*n*) 9
- **firstprivate** (*list*) 9
- **lastprivate** ([*lastprivate-modifier*:] *list*) 9
- **linear** (*list*[: *linear-step*]) 9
- **nowait** 9
- **order** ([*order-modifier* :] **concurrent**) 9
  - *order-modifier*: **reproducible, unconstrained**
- **ordered** [(*n*)]
  - The loops or how many loops to associate with a construct.
- **private** (*list*) 9
- **reduction** 9
- **schedule** ([*modifier* [, *modifier*] :] *kind*[, *chunk_size*])

Values for **schedule** *kind:*

- **static:** Iterations are divided into chunks of size *chunk_size* and assigned to team threads in round-robin fashion in order of thread number.
- **dynamic:** Each thread executes a chunk of iterations then requests another chunk until none remain.
- **guided:** Same as **dynamic**, except chunk size is different for each chunk, with each successive chunk smaller than the last.
- **auto:** Compiler and/or runtime decides.
- **runtime:** Uses *run-sched-var* ICV.

Values for **schedule** *modifier:*

- **monotonic:** Each thread executes its assigned chunks in increasing logical iteration order. A **schedule** (static) clause or **order** clause implies monotonic.
- **nonmonotonic:** Chunks are assigned to threads in any order and the behavior of an application that depends on execution order of the chunks is unspecified.
- **simd:** Ignored when the loop is not associated with a SIMD construct, else *new_chunk_size* for all except the first and last chunks is $\lceil chunk\_size/simd\_width \rceil * simd\_width$ (*simd_width*: implementation-defined value).

distribute [11.6] [2.11.6.1]

Specifies loops which are executed by the initial teams.

C/C++:
```
#pragma omp distribute [clause[ [,]clause] ... ]
  loop-nest
```

Fortran:
```
!$omp distribute [clause[ [,]clause] ... ]
  loop-nest
[!$omp end distribute]
```

clause:

- **allocate** 9
- **collapse** (*n*) 9
- **dist_schedule** (*kind*[, *chunk_size*])
- **firstprivate** (*list*) 9
- **lastprivate** (*list*) 9
- **order** ([*order-modifier* :] **concurrent**) 9
  - *order-modifier*: **reproducible** or **unconstrained**
- **private** (*list*) 9

loop [11.7] [2.11.7]

Specifies that the iterations of the associated loops may execute concurrently and permits the encountering thread(s) to execute the loop accordingly.

C/C++:
```
#pragma omp loop [clause[ [,]clause] ... ]
  loop-nest
```

Fortran:
```
!$omp loop [clause[ [,]clause] ... ]
  loop-nest
[!$omp end loop]
```

clause:

- **bind** (*binding*)
  - *binding*: **teams**, **parallel**, or **thread**.
- **collapse** (*n*) 9
- **lastprivate** (*list*) 9
- **order** ([*order-modifier* :] **concurrent**) 9
  - *order-modifier*: **reproducible** or **unconstrained**
- **private** (*list*) 9
- **reduction** 9

Tasking constructs

task [12.5] [2.12.1]

Defines an explicit task. The data environment of the task is created according to the data-sharing attribute clauses on the **task** construct, per-data environment ICVs, and any defaults that apply.

C/C++:
```
#pragma omp task [clause[ [,]clause] ... ]
  structured-block
```

Fortran:
```
!$omp task [clause[ [,]clause] ... ]
  loosely-structured-block
!$omp end task
- or -
!$omp task [clause[ [,]clause] ...]
  strictly-structured-block
[!$omp end task]
```

clause:

- **affinity** ([*aff-modifier* :] *locator-list*)
  - *aff-modifier*: **iterator**(*iterators-definition*)
- **allocate** 9
- **default** (*data-sharing-attribute*) 9
- **detach** (*event-handle*)
  - Task does not complete until given event is fulfilled. (Also see **omp_fulfilled_event**)
  - *event-handle*:
    - C/C++ type **omp_event_handle_t**
    - For kind **omp_event_handle_kind**
- **if** ([**task** :] *omp-logical-expression*) 9
- **in_reduction** (*reduction-identifier* : *list*) 9
- **final** (*omp-logical-expression*)
  - The generated task will be a final task if the expression evaluates to true.
- **firstprivate** (*list*) 9
- **mergeable**
- **priority** (*priority-value*)
  - Hint to the runtime. Sets max priority value.
- **private** (*list*) 9 **shared** (*list*) 9
- **untied**
  - Task is an untied task, meaning any thread in the team can resume the task region after a suspension.

taskloop [12.6] [2.12.2]

Specifies that the iterations of one or more associated loops will be executed in parallel using OpenMP tasks.

C/C++:
```
#pragma omp taskloop [clause[ [,]clause] ... ]
  loop-nest
```

Fortran:
```
!$omp taskloop [clause[ [,]clause] ... ]
  loop-nest
[!$omp end taskloop ]
```

clause:

- **allocate** 9
- **collapse** (*n*) 9
- **default** (*data-sharing-attribute*) 9
- **final** (*omp-logical-expression*)
  - The generated tasks will be final tasks if the expression evaluates to true.
- **firstprivate** (*list*) 9
- **grainsize** ([**strict** :] *grain-size*)
  - Causes the number of logical loop iterations assigned to each created task to be greater than or equal to the minimum of the value of the *grain-size* expression and the number of logical loop iterations, but less than twice the value of the *grain-size* expression. **strict** forces use of exact grain size, except for last iteration.
- **if** ([**taskloop** :] *omp-logical-expression*) 9
- **in_reduction** (*reduction-identifier* : *list*) 9
- **lastprivate** (*list*) 9
- **mergeable**
- **nogroup**
  - Prevents creation of implicit **taskgroup** region.
- **num_tasks** ([**strict** :] *num-tasks*)
  - Create as many tasks as the minimum of the *num-tasks* expression and the number of logical loop iterations. **strict** forces exactly *num-tasks* tasks to be created.
- **priority** (*priority-value*)
  - Hint to the runtime to set the max priority value. If omitted, priority is zero (lowest).
- **private** (*list*) 9
- **reduction** 9
- **shared** (*list*) 9
- **untied**
  - Generted task is an untied task, meaning any thread in the team can resume the task region after a suspension.

taskyield [12.7] [2.12.4]

Specifies that the current task can be suspended in favor of execution of a different task.

C/C++:
```
#pragma omp taskyield
```

Fortran:
```
!$omp taskyield
```

Device directives and construct

target data [13.5] [2.14.2]

Maps variables to a device data environment for the extent of the region.

C/C++:
```
#pragma omp target data clause[ [ [,]clause] ... ]
  structured-block
```

Fortran:
```
$omp target data clause [ [ [,] clause] ... ]
  loosely-structured-block
!$omp end target data
- or -
!$omp target data clause [ [ [,] clause] ... ]
  strictly-structured-block
[ !$omp end target data]
```

clause:

- **device** (*omp-integer-expression*) 9
- **if** ([**target data** :] *omp-logical-expression*) 9
- **map** ([[*map-modifier*, [*map-modifier*, ...]] *map-type* :] *list*) 9
- **use_device_ptr** (*list*)
- **use_device_addr** (*list*)

Continued

Directives and Constructs (continued)

target enter data [13.6] [2.14.3]

Maps variables to a device data environment.

C/C++	**#pragma omp target enter data** *[clause[[,]clause] ...]*
Fortran	**!$omp target enter data** *[clause[[,]clause] ...]*

clause:

- **depend (***[depend-modifier,] dependence-type* **:** *locator-list***)** 9
- **device (***omp-integer-expression***)** 9
- **if (** *[***target data :** *] omp-logical-expression***)** 9
- **map (***[[map-modifier, [map-modifier, ...]] map-type* **:** *] list***)** 9
- **nowait** 9

target exit data [13.7] [2.14.4]

Unmaps variables from a device data environment.

C/C++	**#pragma omp target exit data** *[clause[[,]clause] ...]*
Fortran	**!$omp target exit data** *[clause[[,]clause] ...]*

clause: Any clause used for **target enter data**. See exception for the **map** clause.

target [13.8] [2.14.5]

Map variables to a device data environment and execute the construct on that device.

```
C/C++:
#pragma omp target [clause[ [,]clause] ... ]
    structured-block

Fortran:
$omp target [clause[ [,]clause] ... ]
    loosely-structured-block
!$omp end target
- or -
!$omp target [clause[ [,]clause] ... ]
    strictly-structured-block
[!$omp end target]
```

clause:

- **allocate** 9
  - *allocator*:
    - C/C++ Identifier of type **omp_allocator_handle_t**
    - For Integer expression of **omp_allocator_handle_kind** *kind*
- **defaultmap (***implicit-behavior[***:** *variable-category]***)**
  - *implicit-behavior*: **alloc**, **default**, **firstprivate**, **from**, **none**, **present**, **to**, **tofrom**
  - *variable-category*: **aggregate**, **all**, **pointer**, **scalar**, For **allocatable**
- **depend (***[depend-modifier,] dependence-type* **:** *locator-list***)** 9
- **device(***[device-modifier***:** *] omp-integer-expression***)** 9
  - *device-modifier*: **ancestor, device_num**
- **firstprivate (***list***)** 9
- **has_device_addr (***list***)**
  - Indicates that *list* items already have device addresses, so may be directly accessed from target device. May include array sections.
- **if (***[* **target :** *] omp-logical-expression***)** 9
- **in_reduction (***reduction-identifier* **:** *list***)** 9
- **is_device_ptr(***list***)**
  - Indicates *list* items are device pointers.
- **map (***[[map-modifier, [map-modifier, ...]] map-type* **:** *] list***)** 9
- **nowait** 9 **private (***list***)** 9
- **thread_limit (***omp-integer-expression***)**
- **uses_allocators (***[[alloc-mod ,] alloc-mod]***:** *allocator***)**
  - Enables the use of each specified allocator in the region associated with the directive.
  - *alloc-mod:*
    - **memspace(** *mem-space-handle* **)**
    - **traits(** *traits-array* **)**
  - *mem-space-handle*:
    - C/C++ Variable of **memspace_handle_t** type
    - For Integer of **memspace_handle_kind** kind
  - *traits-array*: Constant array of traits each of type:
    - C/C++ **omp_alloctrait_t**
    - For **type(omp_alloctrait)**

target update [13.9] [2.14.6]

Makes the corresponding list items in the device data environment consistent with their original list items, according to the specified motion clauses.

C/C++	**#pragma omp target update** *clause[[[,]clause] ...]*
Fortran	**!$omp target update** *clause[[[,]clause] ...]*

clause:

- **nowait** 9
- **depend (***[depend-modifier,] dependence-type* **:** *locator-list***)** 9
- **device (***omp-integer-expression***)** 9
- **from (***[motion-modifier[,] [motion-modifier[,] ...]* **:** *] locator-list***)**
  - *motion-modifier*: **present**, **mapper (***mapper-identifier***)**, **iterator (***iterators-definition***)**
- **if (***[* **target update :** *] omp-logical-expression***)** 9
- **to (***[motion-modifier[,] [motion-modifier[,] ...]* **:** *] locator-list***)**
  - *motion-modifier*: **present**, **mapper (***mapper-identifier***)**, **iterator (***iterators-definition***)**

Interoperability construct

interop [14.1] [2.15.1]

Retrieves interoperability properties from the OpenMP implementation to enable interoperability with foreign execution contexts.

C/C++	**#pragma omp interop** *clause [[[,] clause] ...]*
Fortran	**!$omp interop** *clause [[[,] clause] ...]*

clause:

- **device(***omp-integer-expression***)** 9
- **depend (***[depend-modifier,] dependence-type* **:** *locator-list***)** 9
- **destroy(***interop-var***)**
- **init(***[interop-modifier ,] [interop-type,] interop-type***:** *interop-var***)**
  - *interop-modifier*: **prefer_type(***preference-list***)**
  - *interop-type*: **target**, **targetsync**
  - There can be at most only two *interop-type*.
- **nowait** 9
- **use(***interop-var***)**

Synchronization constructs

critical [15.2] [2.19.1]

Restricts execution of the associated structured block to a single thread at a time.

```
C/C++:
#pragma omp critical [(name) [[,] hint (hint-expression)]]
    structured-block

Fortran:
!$omp critical [(name) [[,] hint (hint-expression)] ]
    loosely-structured-block
!$omp end critical [(name)]
- or -
!$omp critical [(name) [[,] hint (hint-expression)] ]
    strictly-structured-block
[ !$omp end critical [(name)] ]
```

hint-expression:

- **omp_sync_hint_uncontended**
- **omp_sync_hint_contended**
- **omp_sync_hint_speculative**
- **omp_sync_hint_nonspeculative**

barrier [15.3.1] [2.19.2]

Specifies an explicit barrier that prevents any thread in a team from continuing past the barrier until all threads in the team encounter the barrier.

C/C++	**#pragma omp barrier**
Fortran	**!$omp barrier**

taskgroup [15.4] [2.19.6]

Specifies a region which a task cannot leave until all its descendant tasks generated inside the dynamic scope of the region have completed.

```
C/C++:
#pragma omp taskgroup [clause[ [,]clause] ... ]
    structured-block

Fortran:
!$omp taskgroup [clause[ [,]clause] ... ]
    loosely-structured-block
!$omp end taskgroup
- or -
!$omp taskgroup [clause[ [,]clause] ... ]
    strictly-structured-block
[!$omp end taskgroup]
```

clause:

- **allocate** 9
- **task_reduction (***reduction-identifier* **:** *list***)**
  - *reduction-identifier:* See **reduction** 9

taskwait [15.5] [2.19.5]

Specifies a wait on the completion of child tasks of the current task.

C/C++	**#pragma omp taskwait** *[clause[[,] clause] ...]*
Fortran	**!$omp taskwait** *[clause[[,] clause] ...]*

clause:

- **depend (***[depend-modifier,] dependence-type* **:** *locator-list***)** 9
- **nowait** 9

flush [15.8.5] [2.19.8]

Makes a thread's temporary view of memory consistent with memory, and enforces an order on the memory operations of the variables.

C/C++	**#pragma omp flush** *[memory-order-clause]* **[(***list***)]**
Fortran	**!$omp flush** *[memory-order-clause]* **[(***list***)]**

memory-order-clause:

- **seq_cst**, **acq_rel**, **release**, **acquire**, **relaxed**

depobj [15.9.4] [2.19.10.1]

Stand-alone directive that initalizes, updates, or destroys an OpenMP depend object.

C/C++	**#pragma omp depobj (***depend-object***)** *clause*
Fortran	**!$omp depobj (***depend-object***)** *clause*

clause:

- **depend (***dependence-type* **:** *locator***)** 9
- **destroy (***depend-object***)**
- **update (***task-dependence-type***)**
  - Sets the dependence type of an OpenMP depend object to *task-dependence-type*.
  - *task-dependence-type*: **in**, **out**, **inout**, **inoutset**, **mutexinoutset**

Continued

Directives and Constructs (continued)

atomic [15.8.4] [2.19.7]

Ensures a specific storage location is accessed atomically.

C/C++:
```
#pragma omp atomic [clause [ [,] clause] ... ]
  statement
```

Fortran:
```
!$omp atomic [clause[ [ [,] clause] ... ] ]
  statement
[!$omp end atomic]
- or -
!$omp atomic [clause[ [ [,] clause] ... ] [,] ] capture &
    [ [,] clause [ [ [,] clause] ...] ]
  statement
  capture-statement
[!$omp end atomic]
- or -
!$omp atomic [clause[ [ [,] clause] ... ] [,] ] capture &
    [ [,] clause [ [ [,] clause] ...] ]
  capture-statement
  statement
[!$omp end atomic]
```

clause:

atomic-clause: **read, write, update**

memory-order-clause: **seq_cst, acq_rel, release, acquire, relaxed**

extended-atomic: **capture, compare, fail, weak**

capture: Capture the value of the variable being updated atomically.

compare: Perform the atomic update conditionally.

fail (seq_cst | acquire | relaxed): Specify the memory ordering requirements for any comparison performed by any atomic conditional update that fails. Its argument overrides any other specified memory ordering.

weak: Specify that the comparison performed by a conditional atomic update may spuriously fail, evaluating to not equal even when the values are equal.

hint (*hint-expression*)

C/C++ *statement:*

if *atomic clause* is...	*statement*:
read	*v* = *x*;
write	*x* = *expr*;
update	*x*++; *x*--; ++*x*; --*x*; *x binop* = *expr*; *x* = *x binop expr*; *x* = *expr binop x*;
compare is present	*cond-expr-stmt:* *x* = *expr ordop x* ? *expr* : *x*; *x* = *x ordop expr* ? *expr* : *x*; *x* = *x* == *e* ? *d* : *x*; *cond-update-stmt:* **if**(*expr ordop x*) { *x* = *expr*; } **if**(*x ordop expr*) { *x* = *expr*; } **if**(*x* == *e*) { *x* = *d*; }
capture is present	*v* = *expr-stmt* { *v* = *x*; *expr-stmt* } { *expr-stmt v* = *x*; } (*expr-stmt*: *write-expr-stmt*, *update-expr-stmt*, *or cond-expr-stmt*.)
both **compare** and **capture** are present	{ *v* = *x*; *cond-update-stmt* } { *cond-update-stmt v* = *x*; } **if**(*x* == *e*) { *x* = *d*; } **else** { *v* = *x*; } { *r* = *x* == *e*; **if**(*r*) { *x* = *d*; } } { *r* = *x* == *e*; **if**(*r*) { *x* = *d*; } **else** { *v* = *x*; } }

Continued in next column

atomic (continued)

For *capture-statement*: Has the form *v* = *x*

For *statement:*

if *atomic clause* is...	*statement*:
read	*v* = *x*
write	*x* = *expr*
update	*x* = *x operator expr* *x* = *expr operator x* *x* = *intrinsic_procedure_name* (*x*, *expr-list*) *x* = *intrinsic_procedure_name* (*expr-list*, *x*)
intrinsic_procedure_name: **MAX, MIN, IAND, IOR, IEOR**	
operator is one of **+**, *****, **/**, **.AND.**, **.OR.**, **.EQV.**, **.NEQV.**	
if **capture** is present and *statement* is preceded or followed by *capture-statement*	*x* = *expr*, in addition to any other allowed

if **compare** is present	**if** (*x* == *e*) **then**
	x = *d*
	end if
	
	if (*x* == *e*) *x* = *d*

if the **compare** and **capture** clauses are both present, and *statement* is not preceded or followed by *capture-statement*	**if** (*x* == *e*) **then**
	x = *d*
	else *v* = *x*
	end if

ordered [15.10.2] [2.19.9]

Specifies a structured block that is to be executed in loop iteration order in a parallelized loop, or it specifies cross iteration dependences in a doacross loop nest.

C/C++:
```
#pragma omp ordered [clause [[,] clause] ]
  structured-block
- or -
#pragma omp ordered clause[ [ [,] clause] ... ]
```

Fortran:
```
!$omp ordered [clause[ [,] clause] ]
  loosely-structured-block
!$omp end ordered
- or -
!$omp ordered [clause [[,] clause] ]
  strictly-structured-block
[ !$omp end ordered]
- or -
!$omp ordered clause[ [ [,] clause] ... ]
```

clause (for the structured-block forms only):

threads

simd

threads or **simd** indicate the parallelization level with which to associate a construct.

clause (for the standalone forms only):

doacross (*dependence-type* : [*vector*])

Identifies cross-iteration dependences that imply additional constraints on the scheduling of loop iterations.

dependence-type:

source

Specifies the satisfaction of cross-iteration dependences that arise from the current iteration. If **source** is specified, then the *vector* argument is optional; if *vector* is omitted, it is assumed to be **omp_cur_iteration**. At most one **doacross** clause can be specified on a directive with **source** as the *dependence-type*.

sink

Specifies a cross-iteration dependence, where *vector* indicates the iteration that satisfies the dependence. If *vector* does not occur in the iteration space, the **doacross** clause is ignored. If all **doacross** clauses on an ordered construct are ignored then the construct is ignored.

Cancellation constructs

cancel [16.1] [2.20.1]

Activates cancellation of the innermost enclosing region of the type specified.

C/C++:
```
#pragma omp cancel construct-type-clause[ [ , ] \
    if-clause]
```

Fortran:
```
!$omp cancel construct-type-clause[ [ , ] if-clause]
```

if-clause: **if** ([**cancel :**] *omp-logical-expression*)

construct-type-clause:

C/C++ **parallel**, **sections**, **taskgroup**, **for**

For **parallel**, **sections**, **taskgroup**, **do**

cancellation point [16.2] [2.20.2]

Introduces a user-defined cancellation point at which tasks check if cancellation of the innermost enclosing region of the type specified has been activated.

C/C++:
```
#pragma omp cancellation point construct-type-clause
```

Fortran:
```
!$omp cancellation point construct-type-clause
```

construct-type-clause:

parallel
sections
taskgroup
C/C++ **for**
For **do**

Combined Constructs and Directives

The following combined constructs and directives are created following the parameters defined in section 17 of the OpenMP API version 5.2 specification and were explicitly defined in previous versions.

do simd and for simd [17] [2.11.5.2]

Specifies that the iterations of associated loops will be executed in parallel by threads in the team and the iterations executed by each thread can also be executed concurrently using SIMD instructions.

C/C++:
```
#pragma omp for simd [clause[ [,]clause] ... ]
  loop-nest
```

Fortran:
```
!$omp do simd [clause[ [,]clause] ... ]
  loop-nest
[!$omp end do simd [nowait] ]
```

clause: Any of the clauses accepted by the **simd**, **for**, or **do** directives.

distribute simd [17] [2.11.6.2]

Specifies a loop that will be distributed across the primary threads of the teams region and executed concurrently using SIMD instructions.

C/C++:
```
#pragma omp distribute simd [clause[ [,]clause] ... ]
  loop-nest
```

Fortran:
```
!$omp distribute simd [clause[ [,]clause] ... ]
  loop-nest
[!$omp end distribute simd]
```

clause: Any of the clauses accepted by **distribute** or **simd**.

distribute parallel do and distribute parallel for

[17] [2.11.6.3]

Specify a loop that can be executed in parallel by multiple threads that are members of multiple teams.

C/C++:
```
#pragma omp distribute parallel for [clause[ [,]clause] ... ]
  loop-nest
```

Fortran:
```
!$omp distribute parallel do [clause[ [,]clause] ... ]
  loop-nest
[!$omp end distribute parallel do]
```

clause: Any clause used for **distribute**, **parallel for**,or **parallel do**.

Continued

Directives and Constructs (continued)

distribute parallel do simd and distribute parallel for simd [17] [2.11.6.4]

Specifies a loop that can be executed concurrently using SIMD instructions in parallel by multiple threads that are members of multiple teams.

C/C++
```
#pragma omp distribute parallel for simd \
      [clause[ [,]clause] ... ]
   loop-nest
```

Fortran
```
!$omp distribute parallel do simd [clause[ [,]clause] ... ]
   loop-nest
[!$omp end distribute parallel do simd]
```

clause: Any clause used for **distribute**, **parallel for simd**, or **parallel do simd**.

taskloop simd [17] [2.12.3]

Specifies that a loop can be executed concurrently using SIMD instructions, and that those iterations will also be executed in parallel using OpenMP tasks.

C/C++
```
#pragma omp taskloop simd [clause[ [,]clause] ... ]
   loop-nest
```

Fortran
```
!$omp taskloop simd [clause[ [,]clause] ... ]
   loop-nest
[!$omp end taskloop simd]
```

clause: Any clause used for **simd** or **taskloop**.

parallel do and parallel for [17] [2.16.1]

Specifies a **parallel** construct containing a worksharing-loop construct with a canonical loop nest and no other statements.

C/C++
```
#pragma omp parallel for [clause[ [,]clause] ... ]
   loop-nest
```

Fortran
```
!$omp parallel do [clause[ [,]clause] ... ]
   loop-nest
[!$omp end parallel do]
```

clause: Any clause used for **parallel**, **for**, or **do** except the **nowait** clause.

parallel loop [17] [2.16.2]

Shortcut for specifying a **parallel** construct containing a **loop** construct with a canonical loop nest and no other statements.

C/C++
```
#pragma omp parallel loop [clause[ [,]clause] ... ]
   loop-nest
```

Fortran
```
!$omp parallel loop [clause[ [,]clause] ... ]
   loop-nest
[!$omp end parallel loop]
```

clause: Any clause used for **parallel** or **loop**.

parallel sections [17] [2.16.3]

Shortcut for specifying a **parallel** construct containing a **sections** construct and no other statements.

C/C++
```
#pragma omp parallel sections [clause[ [,]clause] ... ]
   {
   [#pragma omp section]
      structured-block-sequence
   [#pragma omp section
      structured-block-sequence]
   ...
   }
```

Fortran
```
!$omp parallel sections [clause[ [,]clause] ... ]
   [!$omp section]
      structured-block-sequence
   [!$omp section
      structured-block-sequence]
   ...
!$omp end parallel sections
```

clause: Any clause used for **parallel** or **sections** [C/C++ except the **nowait** clause].

parallel workshare [17] [2.16.4]

Shortcut for specifying a **parallel** construct containing a **workshare** construct and no other statements.

Fortran
```
$omp parallel workshare [clause[ [,]clause] ... ]
   loosely-structured-block
!$omp end parallel workshare
- or -
!$omp parallel workshare [clause[ [,]clause] ...]
   strictly-structured-block
[!$omp end parallel workshare]
```

clause: Any clause used for **parallel**.

parallel do simd and parallel for simd [17] [2.16.5]

Shortcut for specifying a **parallel** construct containing only one worksharing-loop SIMD construct.

C/C++
```
#pragma omp parallel for simd [clause[ [,]clause] ... ]
   loop-nest
```

Fortran
```
!$omp parallel do simd [clause[ [,]clause] ... ]
   loop-nest
[!$omp end parallel do simd]
```

clause: Any clause used for **parallel**, **for simd**, or **do simd** [C/C++ except the **nowait** clause].

parallel masked [17] [2.16.6]

Shortcut for specifying a **parallel** construct containing a **masked** construct and no other statements.

C/C++
```
#pragma omp parallel masked [clause[ [,]clause] ... ]
    structured-block
```

Fortran
```
$omp parallel masked [clause[ [,]clause] ... ]
   loosely-structured-block
!$omp end parallel masked
- or -
!$omp parallel masked [clause[ [, ]clause] ...]
   strictly-structured-block
[!$omp end parallel masked]
```

clause: Any clause used for **parallel** or **masked**.

masked taskloop [17] [2.16.7]

Shortcut for specifying a **masked** construct containing a **taskloop** construct and no other statements.

C/C++
```
#pragma omp masked taskloop [clause[ [,]clause] ... ]
   loop-nest
```

Fortran
```
!$omp masked taskloop [clause[ [,]clause] ... ]
   loop-nest
[$omp end masked taskloop]
```

clause: Any clause used for **taskloop** or **masked**.

masked taskloop simd [17] [2.16.8]

Shortcut for specifying a **masked** construct containing a **taskloop simd** construct and no other statements.

C/C++
```
#pragma omp masked taskloop simd \
      [clause[ [,]clause] ... ]
   loop-nest
```

Fortran
```
!$omp masked taskloop simd [clause[ [,]clause] ... ]
   loop-nest
[$omp end masked taskloop simd]
```

clause: Any clause used for **masked** or **taskloop simd**.

parallel masked taskloop [17] [2.16.9]

Shortcut for specifying a **parallel** construct containing a **masked taskloop** construct and no other statements.

C/C++
```
#pragma omp parallel masked taskloop \
      [clause[ [,]clause] ... ]
   loop-nest
```

Fortran
```
!$omp parallel masked taskloop [clause[ [,]clause] ... ]
   loop-nest
[$omp end parallel masked taskloop]
```

clause: Any clause used for **parallel** or **masked taskloop** except the **in_reduction** clause.

parallel masked taskloop simd [17] [2.16.10]

Shortcut for specifying a **parallel** construct containing a **masked taskloop simd** construct and no other statements.

C/C++
```
#pragma omp parallel masked taskloop simd \
      [clause[ [,]clause] ... ]
   loop-nest
```

Fortran
```
!$omp parallel masked taskloop simd [clause[ [,] &
      clause] ... ]
   loop-nest
[$omp end parallel masked taskloop simd]
```

clause: Any clause used for **masked taskloop simd** or **parallel** except the **in_reduction** clause.

teams distribute [17] [2.16.11]

Shortcut for specifying a **teams** construct containing a **distribute** construct and no other statements. .

C/C++
```
#pragma omp teams distribute [clause[ [,]clause] ... ]
   loop-nest
```

Fortran
```
!$omp teams distribute [clause[ [,]clause] ... ]
   loop-nest
[!$omp end teams distribute]
```

clause: Any clause used for **teams** or **distribute**.

teams distribute simd [17] [2.16.12]

Shortcut for specifying a **teams** construct containing a **distribute simd** construct and no other statements.

C/C++
```
#pragma omp teams distribute simd \
      [clause[ [,] clause] ... ]
   loop-nest
```

Fortran
```
!$omp teams distribute simd [clause[ [,]clause] ... ]
   loop-nest
[!$omp end teams distribute simd]
```

clause: Any clause used for **teams** or **distribute simd**.

teams distribute parallel do and teams distribute parallel for [17] [2.16.13]

Shortcut for specifying a **teams** construct containing a distribute parallel worksharing-loop construct and no other statements.

C/C++
```
#pragma omp teams distribute parallel for \
      [clause[ [,]clause] ... ]
   loop-nest
```

Fortran
```
!$omp teams distribute parallel do [clause[ [,] &
      clause] ... ]
   loop-nest
[!$omp end teams distribute parallel do]
```

clause: Any clause used for **teams, distribute parallel for**, or **distribute parallel do**.

teams distribute parallel do simd and teams distribute parallel for simd [17] [2.16.14]

Shortcut for specifying a **teams** construct containing a **distribute parallel for simd** or **distribute parallel do simd** construct and no other statements.

C/C++
```
#pragma omp teams distribute parallel for simd \
      [clause[ [,]clause] ... ]
   loop-nest
```

Fortran
```
!$omp teams distribute parallel do simd
      [clause[ [,]clause] ... ]
   loop-nest
[!$omp end teams distribute parallel do simd]
```

clause: Any clause used for **teams, distribute parallel for simd**, or **distribute parallel do simd**.

teams loop [17] [2.16.15]

Shortcut for specifying a **teams** construct containing a **loop** construct and no other statements.

C/C++
```
#pragma omp teams loop [clause[ [,]clause] ... ]
   loop-nest
```

Fortran
```
!$omp teams loop [clause[ [,]clause] ... ]
   loop-nest
[!$omp end teams loop]
```

clause: Any clause used for **teams** or **loop**.

Continued

Directives and Constructs (continued)

target parallel [17] [2.16.16]

Shortcut for specifying a **target** construct containing a **parallel** construct and no other statements.

C/C++:
```
#pragma omp target parallel [clause[ [,]clause] ... ]
    structured-block
```
Fortran:
```
$omp target parallel [clause[ [,]clause] ... ]
    loosely-structured-block
!$omp end target parallel
- or -
!$omp target parallel [clause[ [,]clause] ... ]
    strictly-structured-block
[!$omp end target parallel]
```
clause: Clauses used for **target** or **parallel** except for **copyin**.

target parallel do and target parallel for [17] [2.16.17]

Shortcut for specifying a **target** construct with a parallel worksharing-loop construct and no other statements.

C/C++:
```
#pragma omp target parallel for [clause[ [,] clause] ... ]
    loop-nest
```
Fortran:
```
!$omp target parallel do [clause[ [,]clause] ... ]
    loop-nest
[!$omp end target parallel do]
```
clause: Any clause used for **target**, **parallel for**, or **parallel do**, except for **copyin**.

target parallel do simd and target parallel for simd [17] [2.16.18]

Shortcut for specifying a **target** construct with a parallel worksharing-loop SIMD construct and no other statements.

C/C++:
```
#pragma omp target parallel for simd \
        [clause[ [,]clause] ... ]
    loop-nest
```
Fortran:
```
!$omp target parallel do simd [clause[ [,]clause] ... ]
    loop-nest
[!$omp end target parallel do simd]
```
clause: Any clause used for **target**, **parallel for simd**, or **parallel do simd**, except for **copyin**.

target parallel loop [17] [2.16.19]

Shortcut for specifying a **target** construct containing a **parallel loop** construct and no other statements.

C/C++:
```
#pragma omp target parallel loop [clause[ [,] \
        clause] ... ]
    loop-nest
```
Fortran:
```
!$omp target parallel loop [clause[ [,]clause] ... ]
    loop-nest
[!$omp end target parallel loop]
```
clause: Clauses used for **target** or **parallel loop** except **copyin**.

target simd [17] [2.16.20]

Shortcut for specifying a **target** construct containing a **simd** construct and no other statements.

C/C++:
```
#pragma omp target simd [clause[ [,]clause] ... ]
    loop-nest
```
Fortran:
```
!$omp target simd [clause[ [,]clause] ... ]
    loop-nest
[!$omp end target simd]
```
clause: Any clause used for **target** or **simd**.

target teams [17] [2.16.21]

Shortcut for specifying a **target** construct containing a **teams** construct and no other statements.

C/C++:
```
#pragma omp target teams [clause[ [,]clause] ... ]
    structured-block
```
Fortran:
```
$omp target teams [clause[ [,]clause] ... ]
    loosely-structured-block
!$omp end target teams
- or -
!$omp target teams [clause[ [,]clause] ... ]
    strictly-structured-block
[ !$omp end target teams]
```
clause: Any clause used for **target** or **teams**.

target teams distribute [17] [2.16.22]

Shortcut for specifying a **target** construct containing a **teams distribute** construct and no other statements.

C/C++:
```
#pragma omp target teams distribute [clause[ [,] \
        clause] ... ]
    loop-nest
```
Fortran:
```
!$omp target teams distribute [clause[ [,]clause] ... ]
    loop-nest
[!$omp end target teams distribute]
```
clause: Any clause used for **target** or **teams distribute**.

target teams distribute simd [17] [2.16.23]

Shortcut for specifying a **target** construct containing a **teams distribute simd** construct and no other statements.

C/C++:
```
#pragma omp target teams distribute simd \
        [clause[ [,]clause] ... ]
    loop-nest
```
Fortran:
```
!$omp target teams distribute simd [clause[ [,]clause] ... ]
    loop-nest
[!$omp end target teams distribute simd]
```
clause: Any clause used for **target** or **teams distribute simd**.

target teams loop [17] [2.16.24]

Shortcut for specifying a **target** construct containing a **teams loop** construct and no other statements.

C/C++:
```
#pragma omp target teams loop [clause[ [,]clause] ... ]
    loop-nest
```
Fortran:
```
!$omp target teams loop [clause[ [,]clause] ... ]
    loop-nest
[!$omp end target teams loop]
```
clause: Any clause used for **target** or **teams loop**.

target teams distribute parallel do and target teams distribute parallel for [17] [2.16.25]

Shortcut for specifying a **target** construct containing **teams distribute parallel for**, **teams distribute parallel do** and no other statements.

C/C++:
```
#pragma omp target teams distribute parallel for \
        [clause[ [,]clause] ... ]
    loop-nest
```
Fortran:
```
!$omp target teams distribute parallel do &
        [clause[ [,]clause] ... ]
    loop-nest
[$omp end target teams distribute parallel do]
```
clause:
Any clause used for **target**, **teams distribute parallel for**, or **teams distribute parallel do**.

target teams distribute parallel do simd target teams distribute parallel for simd [17] [2.16.26]

Shortcut for specifying a **target** construct containing a teams distribute parallel worksharing-loop SIMD construct and no other statements.

C/C++:
```
#pragma omp target teams distribute parallel for simd \
        [clause[ [,]clause] ... ]
    loop-nest
```
Fortran:
```
!$omp target teams distribute parallel do simd &
        [clause[ [,]clause] ... ]
    loop-nest
[!$omp end target teams distribute parallel do simd]
```
clause: Any clause used for **target**, **teams distribute parallel for simd**, or **teams distribute parallel do simd**.

Notes

Clauses

All list items appearing in a clause must be visible according to the scoping rules of the base language.

Data sharing attribute clauses [5.4] [2.21.4]

Additional data sharing attribute clauses are **is_device_ptr**, **use_device_ptr**, **has_device_addr**, and **use_device_addr**. These clauses are described at the directives that accept them.

default (shared | firstprivate | private | none)
Default data-sharing attributes are disabled. All variables in a construct must be declared inside the construct or appear in a data-sharing attribute clause.

Used in: parallel (3), task (4), taskloop (4), teams (3)

shared (*list***)**
Variables in *list* are shared between threads or explicit tasks executing the construct.

Used in: parallel (3), task (4), taskloop (4), teams (3)

private (*list***)**
Creates a new variable for each item in *list* that is private to each thread or explicit task. The private variable is not given an initial value.

Used in: distribute (4), do and for (4), loop (4), parallel (3), scope (3), section (4), simd (3), single (3), target (5), task (4), taskloop (4), teams (3)

firstprivate (*list***)**
Declares *list* items to be private to each thread or explicit task and assigns them the value the original variable has at the time the construct is encountered.

Used in: distribute (4), do and for (4), parallel (3), scope (3), section (4), simd (3), target (5), task (4), taskloop (4), teams (3)

lastprivate (*[lastprivate-modifier:] list***)**
After the last loop ends, the variables in *list* will be copied to the primary thread.

lastprivate-modifier: **conditional**

conditional: Uses the value from the thread that executed the highest index iteration number.

Used in: distribute (4), do and for (4), loop (4), section (4), simd (3), taskloop (4)

linear (*linear-list [* **:** *linear-step]* **)**
linear (*linear-list [***:** *linear-modifier [*, *linear-modifier]]***)**
Declares each *linear-list* item to have a linear value or address with respect to the iteration space of the loop.

linear-list: list (or for **declare simd** *argument-list*)

linear-modifier: **step(***linear-step***)**, *linear-type-modifier*

linear-step: OpenMP integer expression (1 is default)

linear-type-modifier: **val**, **ref**, **uval** (**val** is default)

val: The value is linear

ref: The address is linear (C++ and Fortran only)

uval: The value is linear, may not be modified (C++ and Fortran only)

The **ref** and **uval** modifiers may only be specified for a **linear** clause on the **declare simd** directive, and only for arguments that are passed by reference.

Used in: declare simd (2), distribute (4), do and for (4), simd (3)

allocate clause [6.6] [2.13.4]

allocate (*[allocator* **:** *] list***)**
allocate(*allocate-modifier [*, *allocate-modifier]* **:** *list***)**

allocate-modifier:

allocator (*allocator***)**
allocator: is an expression of:
C/C++ type **omp_allocator_handle_t**
For *kind* **omp_allocator_handle_kind**

align (*alignment***)**
alignment: A constant positive integer power of 2.

Used in: distribute (4), do and for (4), parallel (3), scope (3), section (4), single (3), target (5), task (4), taskgroup (5), taskloop (4), teams (3)

collapse clause [4.4.3]

collapse (*n***)**
A constant positive integer expression that specifies how many loops are associated with the construct.

Used in: distribute (4), do and for (4), loop (4), simd (3), taskloop (4)

depend clause [15.9.5] [2.19.11]

Enforces additional constraints on the scheduling of tasks or loop iterations, establishing dependences only between sibling tasks or between loop iterations.

depend (*[depend-modifier,]dependence-type* **:** *locator-list***)**

depend-modifier: **iterator (***iterators-definition***)**

dependence-type: **in**, **out**, **inout**, **mutexinoutset**, **inoutset**, **depobj**

- **in**: The generated task will be a dependent task of all previously generated sibling tasks that reference at least one of the list items in an **out** or **inout** *dependence-type* list.
- **out** and **inout**: The generated task will be a dependent task of all previously generated sibling tasks that reference at least one of the list items in an **in**, **out**, **mutexinoutset**, **inout**, or **inoutset** *dependence-type* list.
- **mutexinoutset**: If the storage location of at least one of the list items is the same as that of a list item appearing in a **depend** clause with an **in**, **out**, **inout**, or **inoutset** *dependence-type* on a construct from which a sibling task was previously generated, then the generated task will be a dependent task of that sibling task. If the storage location of at least one of the list items is the same as that of a list item appearing in a **depend** clause with a **mutexinoutset** *dependence-type* on a construct from which a sibling task was previously generated, then the sibling tasks will be mutually exclusive tasks.
- **inoutset**: If the storage location of at least one of the list items matches the storage location of a list item appearing in a **depend** clause with an **in**, **out**, **inout**, or **mutexinoutset** *dependence-type* on a construct from which a sibling task was previously generated, then the generated task will be a dependent task of that sibling task.
- **depobj**: The task dependences are derived from the **depend** clause specified in the **depobj** constructs that initialized dependences represented by the depend objects specified in the **depend** clause as if the **depend** clauses of the **depobj** constructs were specified in the current construct.

Used in: depobj (5), dispatch (2), interop (5), target (5), target enter data (5), target exit data (5), target update (5), task (4), taskwait (5)

device clause [13.2]

device (*[***ancestor | device_num :** *] device-description***)**
Identifies the target device that is associated with a device construct.

device-description: An expression of type integer that refers to the device number or, if **ancestor** modifier is specified, must be 1.

Used in: dispatch (2), interop (5), target (5), target data (4), target enter data (5), target exit data (5), target update (5)

if clause [3.4] [2.18]

The effect of the **if** clause depends on the construct to which it is applied. For combined or composite constructs, it only applies to the semantics of the construct named in the *directive-name-modifier* if one is specified. If no modifier is specified for a combined or composite construct then the **if** clause applies to all constructs to which an **if** clause can apply.

if (*[directive-name-modifier* **:** *] omp-logical-expression***)**

Used in: cancel (6), parallel (3), simd (3), target (5), target data (4), target enter data (5), target exit data (5), target update (5), task (4), taskloop (4), teams (3)

map clause [5.8.3] [2.21.7.1]

map (*[[map-modifier*, *[map-modifier*, *...] map-type* **:** *] locator-list***)**
Maps data from the task's environment to the device environment.

map-type: **alloc**, **to**, **from**, **tofrom**, **release**, **delete**

For the **target** or **target data** directives:
map-type: **alloc**, **to**, **from**, **tofrom**, **release**

For the **target enter data** directive:
map-type: **alloc**, **to**, **from**, **tofrom**

For the **target exit data** directive:
map-type: **to**, **from**, **tofrom**, **release**, **delete**

map-modifier: **always**, **close**, **present**, **mapper(***mapper-id***)**, **iterator(***iterators-definition***)**

Used in: declare mapper (1), target (5), target data (4), target enter data (5), target exit data (5)

order clause [10.3] [2.11.3]

order (*[order-modifier* **:** *]* **concurrent)**

order-modifier: **reproducible**, **unconstrained**

Specifies an expected order of execution for the iterations of the associated loops of a loop-associated directive.

Used in: distribute (4), do and for (4), loop (4), simd (3)

nowait clause [15.6]

nowait
Overrides any synchronization that would otherwise occur at the end of a construct. It can also specify that an *interoperability requirement set* includes the *nowait* property. If the construct includes an implicit barrier, the **nowait** clause specifies that the barrier will not occur.

Used in: dispatch (2), do and for (4), interop (5), scope (3), section (4), single (3), target (5), target enter data (5), target exit data (5), target update (5), taskwait (5), workshare (3)

reduction clause [5.5.8] [2.21.5.4]

reduction (*[reduction-modifier* **,** *] reduction-identifier* **:** *list***)**
Specifies a *reduction-identifier* and one or more list items.

reduction-modifier: **inscan**, **task**, **default**

C++ *reduction-identifier*:
Either an *id-expression* or one of the following operators: **+**, *****, **&**, **|**, **^**, **&&**, **||**

C *reduction-identifier*:
Either an *identifier* or one of the following operators: **+**, *****, **&**, **|**, **^**, **&&**, **||**

For *reduction-identifier*:
Either a base language identifier, a user-defined operator, one of the following operators: **+**, *****, **.and.**, **.or.**, **.eqv.**, **.neqv.**, or one of the following intrinsic procedure names: **max**, **min**, **iand**, **ior**, **ieor**.

Used in: do and for (4), loop (4), parallel (3), scope (3), section (4), simd (3), taskloop (4), teams (3)

iterator [3.2.6] [2.1.6]

Identifiers that expand to multiple values in the clause on which they appear.

iterator (*iterators-definition***)**

iterators-definition:
iterator-specifier [, *iterators-definition]*

iterators-specifier:
[iterator-type] identifier **=** *range-specification*

identifier: A base language identifier.

range-specification: *begin* **:** *end[***:** *step]*
begin, *end*: Expressions for which their types can be converted to *iterator-type*

step: An integral expression.

iterator-type: C/C++ A type name. For A type specifier.

Runtime Library Routines

Thread team routines

omp_set_num_threads [18.2.1] [3.2.1]

Affects the number of threads used for subsequent **parallel** constructs not specifying a **num_threads** clause, by setting the value of the first element of the *nthreads-var* ICV of the current task to *num_threads*.

C/C++	**void omp_set_num_threads (int** *num_threads***);**
Fortran	**subroutine omp_set_num_threads (***num_threads***)** **integer** *num_threads*

omp_get_num_threads [18.2.2] [3.2.2]

Returns the number of threads in the current team. The binding region for an **omp_get_num_threads** region is the innermost enclosing **parallel** region. If called from the sequential part of a program, this routine returns 1.

C/C++	**int omp_get_num_threads (void);**
Fortran	**integer function omp_get_num_threads ()**

omp_get_max_threads [18.2.3] [3.2.3]

Returns an upper bound on the number of threads that could be used to form a new team if a **parallel** construct without a **num_threads** clause were encountered after execution returns from this routine.

C/C++	**int omp_get_max_threads (void);**
Fortran	**integer function omp_get_max_threads ()**

omp_get_thread_num [18.2.4] [3.2.4]

Returns the thread number of the calling thread, within the current team.

C/C++	**int omp_get_thread_num (void);**
Fortran	**integer function omp_get_thread_num ()**

omp_in_parallel [18.2.5] [3.2.5]

Returns *true* if the *active-levels-var* ICV is greater than zero; otherwise it returns *false*.

C/C++	**int omp_in_parallel (void);**
Fortran	**logical function omp_in_parallel ()**

omp_set_dynamic [18.2.6] [3.2.6]

Enables or disables dynamic adjustment of the number of threads available for the execution of subsequent **parallel** regions by setting the value of the *dyn-var* ICV.

C/C++	**void omp_set_dynamic (int** *dynamic_threads***);**
Fortran	**subroutine omp_set_dynamic (***dynamic_threads***)** **logical** *dynamic_threads*

omp_get_dynamic [18.2.7] [3.2.7]

Returns *true* if dynamic adjustment of the number of threads is enabled for the current task. ICV: *dyn-var*

C/C++	**int omp_get_dynamic (void);**
Fortran	**logical function omp_get_dynamic ()**

omp_get_cancellation [18.2.8] [3.2.8]

Returns *true* if cancellation is enabled; otherwise it returns *false*. ICV: *cancel-var*

C/C++	**int omp_get_cancellation (void);**
Fortran	**logical function omp_get_cancellation ()**

omp_set_schedule [18.2.11] [3.2.11]

Affects the schedule that is applied when **runtime** is used as schedule kind, by setting the value of the *run-sched-var* ICV.

C/C++	**void omp_set_schedule(omp_sched_t** *kind*, **int** *chunk_size***);**
Fortran	**subroutine omp_set_schedule (***kind*, *chunk_size***)** **integer (kind=omp_sched_kind)** *kind* **integer** *chunk_size*

See **omp_get_schedule** for *kind*.

omp_get_schedule [18.2.12] [3.2.12]

Returns the schedule applied when **runtime** schedule is used. ICV: *run-sched-var*

C/C++	**void omp_get_schedule (** **omp_sched_t ****kind*, **int ****chunk_size***);**
Fortran	**subroutine omp_get_schedule (***kind*, *chunk_size***)** **integer (kind=omp_sched_kind)** *kind* **integer** *chunk_size*

kind for **omp_set_schedule** and **omp_get_schedule** is an implementation-defined schedule or:

- **omp_sched_static**
- **omp_sched_dynamic**
- **omp_sched_guided**
- **omp_sched_auto**

Use + or | operators (*C/C++*) or the + operator (*For*) to combine the *kinds* with the modifier **omp_sched_monotonic**.

omp_get_thread_limit [18.2.13] [3.2.13]

Returns the maximum number of OpenMP threads available in contention group. ICV: *thread-limit-var*

C/C++	**int omp_get_thread_limit (void);**
Fortran	**integer function omp_get_thread_limit ()**

omp_get_supported_active_levels [18.2.14] [3.2.14]

Returns the number of active levels of parallelism supported.

C/C++	**int omp_get_supported_active_levels (void);**
Fortran	**integer function omp_get_supported_active_levels ()**

omp_set_max_active_levels [18.2.15] [3.2.15]

Limits the number of nested active parallel regions when a new nested parallel region is generated by the current task, by setting *max-active-levels-var* ICV.

C/C++	**void omp_set_max_active_levels (int** *max_levels***);**
Fortran	**subroutine omp_set_max_active_levels (***max_levels***)** **integer** *max_levels*

omp_get_max_active_levels [18.2.16] [3.2.16]

Returns the maximum number of nested active parallel regions when the innermost parallel region is generated by the current task. ICV: *max-active-levels-var*

C/C++	**int omp_get_max_active_levels (void);**
Fortran	**integer function omp_get_max_active_levels ()**

omp_get_level [18.2.17] [3.2.17]

Returns the number of nested parallel regions on the device that enclose the task containing the call. ICV: *levels-var*

C/C++	**int omp_get_level (void);**
Fortran	**integer function omp_get_level ()**

omp_get_ancestor_thread_num [18.2.18] [3.2.18]

Returns, for a given nested level of the current thread, the thread number of the ancestor of the current thread.

C/C++	**int omp_get_ancestor_thread_num (int** *level***);**
Fortran	**integer function omp_get_ancestor_thread_num (***level***)** **integer** *level*

omp_get_team_size [18.2.19] [3.2.19]

Returns, for a given nested level of the current thread, the size of the thread team to which the ancestor or the current thread belongs.

C/C++	**int omp_get_team_size (int** *level***);**
Fortran	**integer function omp_get_team_size (***level***)** **integer** *level*

omp_get_active_level [18.2.20] [3.2.20]

Returns the number of active, nested parallel regions on the device enclosing the task containing the call. ICV: *active-level-var*

C/C++	**int omp_get_active_level (void);**
Fortran	**integer function omp_get_active_level ()**

Thread affinity routines

omp_get_proc_bind [18.3.1] [3.3.1]

Returns the thread affinity policy to be used for the subsequent nested **parallel** regions that do not specify a **proc_bind** clause.

C/C++	**omp_proc_bind_t omp_get_proc_bind (void);**
Fortran	**integer (kind=omp_proc_bind_kind) &** **function omp_get_proc_bind ()**

Valid return values include:

- **omp_proc_bind_false**
- **omp_proc_bind_true**
- **omp_proc_bind_primary**
- **omp_proc_bind_close**
- **omp_proc_bind_spread**

omp_get_num_places [18.3.2] [3.3.2]

Returns the number of places available to the execution environment in the place list.

C/C++	**int omp_get_num_places (void);**
Fortran	**integer function omp_get_num_places ()**

omp_get_place_num_procs [18.3.3] [3.3.3]

Returns the number of processors available to the execution environment in the specified place.

C/C++	**int omp_get_place_num_procs (int** *place_num***);**
Fortran	**integer function &** **omp_get_place_num_procs (***place_num***)** **integer** *place_num*

omp_get_place_proc_ids [18.3.4] [3.3.4]

Returns numerical identifiers of the processors available to the execution environment in the specified place.

C/C++	**void omp_get_place_proc_ids (** **int** *place_num*, **int ****ids***);**
Fortran	**subroutine omp_get_place_proc_ids(***place_num*, *ids***)** **integer** *place_num* **integer** *ids* **(*)**

omp_get_place_num [18.3.5] [3.3.5]

Returns the place number of the place to which the encountering thread is bound.

C/C++	**int omp_get_place_num (void);**
Fortran	**integer function omp_get_place_num ()**

Continued

Runtime Library Routines (continued)

omp_get_partition_num_places [18.3.6] [3.3.6]

Returns the number of places in the *place-partition-var* ICV of the innermost implicit task.

C/C++
```
int omp_get_partition_num_places (void);
```
Fortran
```
integer function omp_get_partition_num_places ()
```

omp_get_partition_place_nums [18.3.7] [3.3.7]

Returns the list of place numbers corresponding to the places in the *place-partition-var* ICV of the innermost implicit task.

C/C++
```
void omp_get_partition_place_nums (
    int *place_nums);
```
Fortran
```
subroutine omp_get_partition_place_nums( &
    place_nums)
integer place_nums (*)
```

omp_set_affinity_format [18.3.8] [3.3.8]

Sets the affinity format to be used on the device by setting the value of the *affinity-format-var* ICV.

C/C++
```
void omp_set_affinity_format (const char *format);
```
Fortran
```
subroutine omp_set_affinity_format (format)
character(len=*), intent(in) :: format
```

omp_get_affinity_format [18.3.9] [3.3.9]

Returns the value of the *affinity-format-var* ICV on the device.

C/C++
```
size_t omp_get_affinity_format (char *buffer,
    size_t size);
```
Fortran
```
integer function omp_get_affinity_format (buffer)
character(len=*), intent(out) :: buffer
```

omp_display_affinity [18.3.10] [3.3.10]

Prints the OpenMP thread affinity information using the format specification provided.

C/C++
```
void omp_display_affinity (const char *format);
```
Fortran
```
subroutine omp_display_affinity (format)
character(len=*), intent(in) :: format
```

omp_capture_affinity [18.3.11] [3.3.11]

Prints the OpenMP thread affinity information into a buffer using the format specification provided.

C/C++
```
size_t omp_capture_affinity (char *buffer, size_t size,
    const char *format)
```
Fortran
```
integer function omp_capture_affinity (buffer, format)
character(len=*), intent(out) :: buffer
character(len=*), intent(in) :: format
```

Teams region routines

omp_get_num_teams [18.4.1] [3.4.1]

Returns the number of initial teams in the current **teams** region.

C/C++
```
int omp_get_num_teams (void);
```
Fortran
```
integer function omp_get_num_teams ()
```

omp_get_team_num [18.4.2] [3.4.2]

Returns the initial team number of the calling thread.

C/C++
```
int omp_get_team_num (void);
```
Fortran
```
integer function omp_get_team_num ()
```

omp_set_num_teams [18.4.3] [3.4.3]

Sets the value of the *nteams-var* ICV of the current device, affecting the number of threads to be used for subsequent **teams** regions that do not specify a **num_teams** clause.

C/C++
```
void omp_set_num_teams (int num_teams);
```
Fortran
```
subroutine omp_set_num_teams(num_teams)
integer num_teams
```

omp_get_max_teams [18.4.4] [3.4.4]

Returns an upper bound on the number of teams that could be created by a **teams** construct without a **num_teams** clause that is encountered after execution returns from this routine. ICV: *nteams-var*

C/C++
```
int omp_get_max_teams (void);
```
Fortran
```
integer function omp_get_max_teams()
```

omp_set_teams_thread_limit [18.4.5] [3.4.5]

Sets the maximum number of OpenMP threads that can participate in each contention group created by a **teams** construct by setting the value of *teams-thread-limit-var* ICV.

C/C++
```
void omp_set_teams_thread_limit(int thread_limit);
```
Fortran
```
subroutine &
    omp_set_teams_thread_limit(thread_limit)
integer thread_limit
```

omp_get_teams_thread_limit [18.4.6] [3.4.6]

Returns the maximum number of OpenMP threads available to participate in each contention group created by a **teams** construct.

C/C++
```
int omp_get_teams_thread_limit (void);
```
Fortran
```
integer function omp_get_teams_thread_limit ()
```

Tasking routines

omp_get_max_task_priority [18.5.1] [3.5.1]

Returns the maximum value that can be specified in the **priority** clause.

C/C++
```
int omp_get_max_task_priority (void);
```
Fortran
```
integer function omp_get_max_task_priority ()
```

omp_in_final [18.5.3] [3.5.2]

Returns *true* if the routine is executed in a final task region; otherwise, it returns *false*.

C/C++
```
int omp_in_final (void);
```
Fortran
```
logical function omp_in_final ()
```

Resource relinquishing routines

omp_pause_resource [18.6.1] [3.6.1]
omp_pause_resource_all [18.6.2] [3.6.2]

Allows the runtime to relinquish resources used by OpenMP on the specified device. Valid *kind* values include **omp_pause_soft** and **omp_pause_hard**.

C/C++
```
int omp_pause_resource (
    omp_pause_resource_t kind, int device_num);

int omp_pause_resource_all (
    omp_pause_resource_t kind);
```
Fortran
```
integer function omp_pause_resource (  &
    kind, device_num)
integer (kind=omp_pause_resource_kind) kind
integer device_num

integer function omp_pause_resource_all (kind)
integer (kind=omp_pause_resource_kind) kind
```

Device information routines

omp_get_num_procs [18.7.1] [3.7.1]

Returns the number of processors that are available to the device at the time the routine is called.

C/C++
```
int omp_get_num_procs (void);
```
Fortran
```
integer function omp_get_num_procs ()
```

omp_set_default_device [18.7.2] [3.7.2]

Assigns the value of the *default-device-var* ICV, which determines default target device.

C/C++
```
void omp_set_default_device (int device_num);
```
Fortran
```
subroutine omp_set_default_device (device_num)
integer device_num
```

omp_get_default_device [18.7.3] [3.7.3]

Returns the value of the *default-device-var* ICV, which determines the default target device.

C/C++
```
int omp_get_default_device (void);
```
Fortran
```
integer function omp_get_default_device ()
```

omp_get_num_devices [18.7.4] [3.7.4]

Returns the number of non-host devices available for offloading code or data.

C/C++
```
int omp_get_num_devices (void);
```
Fortran
```
integer function omp_get_num_devices ()
```

omp_get_device_num [18.7.5] [3.7.5]

Returns the device number of the device on which the calling thread is executing.

C/C++
```
int omp_get_device_num (void);
```
Fortran
```
integer function omp_get_device_num ()
```

omp_is_initial_device [18.7.6] [3.7.6]

Returns *true* if the current task is executing on the host device; otherwise, it returns *false*.

C/C++
```
int omp_is_initial_device (void);
```
Fortran
```
logical function omp_is_initial_device ()
```

omp_get_initial_device [18.7.7] [3.7.7]

Returns a device number representing the host device.

C/C++
```
int omp_get_initial_device (void);
```
Fortran
```
integer function omp_get_initial_device()
```

Device memory routines

These routines support allocation and management of pointers in the data environments of target devices.

omp_target_alloc [18.8.1] [3.8.1]

Allocates memory in a device data environment and returns a device pointer to that memory.

C/C++
```
void *omp_target_alloc (size_t size, int device_num);
```
Fortran
```
type(c_ptr) function omp_target_alloc( &
    size, device_num) bind(c)
use, intrinsic :: iso_c_binding, only : c_ptr, &
    c_size_t, c_int
integer(c_size_t), value :: size
integer(c_int), value :: device_num
```

omp_target_free [18.8.2] [3.8.2]

Frees the device memory allocated by the **omp_target_alloc** routine.

C/C++
```
void omp_target_free (void *device_ptr,
    int device_num);
```
Fortran
```
subroutine omp_target_free(device_ptr, device_num) &
    bind(c)
use, intrinsic :: iso_c_binding, only : c_ptr, c_int
type(c_ptr), value :: device_ptr
integer(c_int), value :: device_num
```

Continued

Runtime Library Routines (continued)

omp_target_is_present [18.8.3] [3.8.3]

Tests whether a host pointer refers to storage that is mapped to a given device.

C/C++:
```
int omp_target_is_present (const void *ptr,
    int device_num);
```

Fortran:
```
integer(c_int) function omp_target_is_present( &
    ptr, device_num) bind(c)
use, intrinsic :: iso_c_binding, only : c_ptr, c_int
type(c_ptr), value :: ptr
integer(c_int), value :: device_num
```

omp_target_is_accessible [18.8.4] [3.8.4]

Tests whether host memory is accessible from a given device.

C/C++:
```
int omp_target_is_accessible (const void *ptr,
    size_t size, int device_num);
```

Fortran:
```
integer(c_int) function omp_target_is_accessible( &
    ptr, size, device_num) bind(c)
use, intrinsic :: iso_c_binding, only : &
    c_ptr, c_size_t, c_int
type(c_ptr), value :: ptr
integer(c_size_t), value :: size
integer(c_int), value :: device_num
```

omp_target_memcpy [18.8.5] [3.8.5]

Copies memory between any combination of host and device pointers.

C/C++:
```
int omp_target_memcpy (void *dst, const void *src,
    size_t length, size_t dst_offset, size_t src_offset,
    int dst_device_num, int src_device_num);
```

Fortran:
```
integer(c_int) function omp_target_memcpy( &
    dst, src, length, dst_offset, src_offset, &
    dst_device_num, src_device_num) bind(c)
use, intrinsic :: iso_c_binding, only : c_ptr, &
    c_int, c_size_t
type(c_ptr), value :: dst, src
integer(c_size_t), value :: length, dst_offset, src_offset
integer(c_int), value :: dst_device_num, &
    src_device_num
```

omp_target_memcpy_rect [18.8.6] [3.8.6]

Copies a rectangular subvolume from a multi-dimensional array to another multi-dimensional array.

C/C++:
```
int omp_target_memcpy_rect (void *dst,
    const void *src, size_t element_size, int num_dims,
    const size_t *volume, const size_t *dst_offsets,
    const size_t *src_offsets,
    const size_t *dst_dimensions,
    const size_t *src_dimensions, int dst_device_num,
    int src_device_num);
```

Fortran:
```
integer(c_int) function omp_target_memcpy_rect( &
    dst, src ,element_size, num_dims, volume, &
    dst_offsets, src_offsets, dst_dimensions, &
    src_dimensions, dst_device_num, &
    src_device_num) bind(c)
use, intrinsic :: iso_c_binding, only : c_ptr, c_int, &
    c_size_t
type(c_ptr), value :: dst, src
integer(c_size_t), value :: element_size
integer(c_int), value :: num_dims, dst_device_num, &
    src_device_num
integer(c_size_t), intent(in) :: volume(*), dst_offsets(*), &
src_offsets(*), dst_dimensions(*), src_dimensions(*)
```

omp_target_memcpy_async [18.8.7] [3.8.7]

Performs a copy between any combination of host and device pointers asynchronously.

C/C++:
```
int omp_target_memcpy_async (void *dst,
    const void *src, size_t length, size_t dst_offset,
    size_t src_offset, int dst_device_num,
    int src_device_num, int depobj_count,
    omp_depend_t *depobj_list);
```

Fortran:
```
integer(c_int) function omp_target_memcpy_async( &
    dst, src, length, dst_offset, src_offset, &
    dst_device_num, src_device_num, depobj_count, &
    depobj_list) bind(c)
use, intrinsic :: iso_c_binding, only : c_ptr, c_int, &
    c_size_t
type(c_ptr), value :: dst, src
integer(c_size_t), value :: length, dst_offset, src_offset
integer(c_int), value :: dst_device_num, &
    src_device_num, depobj_count
integer(omp_depend_kind), optional :: depobj_list(*)
```

omp_target_memcpy_rect_async [18.8.8] [3.8.8]

Asynchronously performs a copy between any combination of host and device pointers.

C/C++:
```
int omp_target_memcpy_rect_async (void *dst,
    const void *src, size_t element_size, int num_dims,
    const size_t *volume, const size_t *dst_offsets,
    const size_t *src_offsets, const size_t *dst_dimensions,
    const size_t *src_dimensions, int dst_device_num,
    int src_device_num, int depobj_count,
    omp_depend_t *depobj_list);
```

Fortran:
```
integer(c_int) function &
    omp_target_memcpy_rect_async ( &
    dst, src, element_size, num_dims, volume, &
    dst_offsets, src_offsets, dst_dimensions, &
    src_dimensions, dst_device_num, src_device_num, &
    depobj_count, depobj_list) bind(c)
use, intrinsic :: iso_c_binding, only : c_ptr, c_int, &
    c_size_t
type(c_ptr), value :: dst, src
integer(c_size_t), value :: element_size
integer(c_int), value :: num_dims, dst_device_num, &
    src_device_num, depobj_count
integer(c_size_t), intent(in) :: volume(*), dst_offsets(*), &
    src_offsets(*), dst_dimensions(*), src_dimensions(*)
integer(omp_depobj_kind), optional :: depobj_list(*)
```

omp_target_associate_ptr [18.8.9] [3.8.9]

Maps a device pointer, which may be returned from **omp_target_alloc** or implementation-defined runtime routines, to a host pointer.

C/C++:
```
int omp_target_associate_ptr (const void *host_ptr,
    const void *device_ptr, size_t size,
    size_t device_offset, int device_num);
```

Fortran:
```
integer(c_int) function omp_target_associate_ptr( &
    host_ptr, device_ptr, size, device_offset, &
    device_num) bind(c)
use, intrinsic :: iso_c_binding, only : c_ptr, &
    c_size_t, c_int
type(c_ptr), value :: host_ptr, device_ptr
integer(c_size_t), value :: size, device_offset
integer(c_int), value :: device_num
```

omp_target_disassociate_ptr [18.8.10] [3.8.10]

Removes the association between a host pointer and a device address on a given device.

C/C++:
```
int omp_target_disassociate_ptr (const void *ptr,
    int device_num);
```

Fortran:
```
integer(c_int) function omp_target_disassociate_ptr(&
    ptr, device_num)  bind(c)
use, intrinsic :: iso_c_binding, only : c_ptr, c_int
type(c_ptr), value :: ptr
integer(c_int), value :: device_num
```

omp_get_mapped_ptr [18.8.11] [3.8.11]

Returns the device pointer that is associated with a host pointer for a given device.

C/C++:
```
void *omp_get_mapped_ptr (const void *ptr,
    int device_num);
```

Fortran:
```
type(c_ptr) function omp_get_mapped_ptr( &
    ptr, device_num) bind(c)
use, intrinsic :: iso_c_binding, only : c_ptr, c_int
type(c_ptr), value :: ptr
integer(c_int), value :: device_num
```

Lock routines

General-purpose lock routines. Two types of locks are supported: simple locks and nestable locks. A nestable lock can be set multiple times by the same task before being unset; a simple lock cannot be set if it is already owned by the task trying to set it.

Initialize lock [18.9.1] [3.9.1]

C/C++:
```
void omp_init_lock (omp_lock_t *lock);
void omp_init_nest_lock (omp_nest_lock_t *lock);
```

Fortran:
```
subroutine omp_init_lock (svar)
integer (kind=omp_lock_kind) svar
subroutine omp_init_nest_lock (nvar)
integer (kind=omp_nest_lock_kind) nvar
```

Initialize lock with hint [18.9.2] [3.9.2]

C/C++:
```
void omp_init_lock_with_hint (
    omp_lock_t *lock,
    omp_sync_hint_t hint);
void omp_init_nest_lock_with_hint (
    omp_nest_lock_t *lock,
    omp_sync_hint_t hint);
```

Fortran:
```
subroutine omp_init_lock_with_hint (svar, hint)
integer (kind=omp_lock_kind) svar
integer (kind=omp_sync_hint_kind) hint
subroutine omp_init_nest_lock_with_hint (nvar, hint)
integer (kind=omp_nest_lock_kind) nvar
integer (kind=omp_sync_hint_kind) hint
```

hint: See [15.1][2.19.12] in the specification.

Destroy lock [18.9.3] [3.9.3]

Ensure that the OpenMP lock is uninitialized.

C/C++:
```
void omp_destroy_lock (omp_lock_t *lock);
void omp_destroy_nest_lock (omp_nest_lock_t *lock);
```

Fortran:
```
subroutine omp_destroy_lock (svar)
integer (kind=omp_lock_kind) svar
subroutine omp_destroy_nest_lock (nvar)
integer (kind=omp_nest_lock_kind) nvar
```

Set lock [18.9.4] [3.9.4]

Sets an OpenMP lock. The calling task region is suspended until the lock is set.

C/C++:
```
void omp_set_lock (omp_lock_t *lock);
void omp_set_nest_lock (omp_nest_lock_t *lock);
```

Fortran:
```
subroutine omp_set_lock (svar)
integer (kind=omp_lock_kind) svar
subroutine omp_set_nest_lock (nvar)
integer (kind=omp_nest_lock_kind) nvar
```

Unset lock [18.9.5] [3.9.5]

C/C++:
```
void omp_unset_lock (omp_lock_t *lock);
void omp_unset_nest_lock (omp_nest_lock_t *lock);
```

Fortran:
```
subroutine omp_unset_lock (svar)
integer (kind=omp_lock_kind) svar
subroutine omp_unset_nest_lock (nvar)
integer (kind=omp_nest_lock_kind) nvar
```

Continued

Runtime Library Routines (continued)

Test lock [18.9.6] [3.9.6]

Attempt to set an OpenMP lock but do not suspend execution of the task executing the routine.

C/C++
```
int omp_test_lock (omp_lock_t *lock);
int omp_test_nest_lock (omp_nest_lock_t *lock);
```

Fortran
```
logical function omp_test_lock (svar)
integer (kind=omp_lock_kind) svar
integer function omp_test_nest_lock (nvar)
integer (kind=omp_nest_lock_kind) nvar
```

Timing routines

Timing routines support a portable wall clock timer. These record elapsed time per-thread and are not guaranteed to be globally consistent across all the threads participating in an application.

omp_get_wtime [18.10.1] [3.10.1]

Returns elapsed wall clock time in seconds.

C/C++
```
double omp_get_wtime (void);
```

Fortran
```
double precision function omp_get_wtime ()
```

omp_get_wtick [18.10.2] [3.10.2]

Returns the precision of the timer (seconds between ticks) used by **omp_get_wtime**.

C/C++
```
double omp_get_wtick (void);
```

Fortran
```
double precision function omp_get_wtick ()
```

Event routine

Event routines support OpenMP event objects, which must be accessed through the routines described in this section or through the **detach** clause.

omp_fulfill_event [18.11.1] [3.11.1]

Fulfills and destroys an OpenMP event.

C/C++
```
void omp_fulfill_event (omp_event_handle_t event);
```

Fortran
```
subroutine omp_fulfill_event (event)
integer (kind=omp_event_handle_kind) event
```

Interoperability routines

omp_get_num_interop_properties [18.12.1] [3.12.1]

Retrieves the number of implementation-defined properties available for an **omp_interop_t** object.

C/C++
```
int omp_get_num_interop_properties (
    omp_interop_t interop);
```

omp_get_interop_int [18.12.2] [3.12.2]

Retrieves an integer property from an **omp_interop_t** object.

C/C++
```
omp_intptr_t omp_get_interop_int (
    const omp_interop_t interop,
    omp_interop_property_t property_id,
    int *ret_code );
```

omp_get_interop_ptr [18.12.3] [3.12.3]

Retrieves a pointer property from an **omp_interop_t** object.

C/C++
```
void *omp_get_interop_ptr (
    const omp_interop_t interop,
    omp_interop_property_t property_id,
    int *ret_code );
```

omp_get_interop_str [18.12.4] [3.12.4]

Retrieves a string property from an **omp_interop_t** object.

C/C++
```
const char* omp_get_interop_str (
    const omp_interop_t interop,
    omp_interop_property_t property_id,
    int *ret_code );
```

omp_get_interop_name [18.12.5] [3.12.5]

Retrieves a property name from an **omp_interop_t** object.

C/C++
```
const char* omp_get_interop_name (
    omp_interop_t interop,
    omp_interop_property_t property_id);
```

omp_get_interop_type_desc [18.12.6] [3.12.6]

Retrieves a description of the type of a property associated with an **omp_interop_t** object.

C/C++
```
const char* omp_get_interop_type_desc (
    omp_interop_t interop,
    omp_interop_property_t property_id);
```

omp_get_interop_rc_desc [18.12.7] [3.12.7]

Retrieves a description of the return code associated with an **omp_interop_t** object.

C/C++
```
const char* omp_get_interop_rc_desc (
    omp_interop_t ret_code);
```

Memory management routines

Memory Management Types [18.13.1] [3.13.1]

The **omp_alloctrait_t** struct in C/C++ and **omp_alloctrait** type in Fortran define members named *key* and *value*, with these types and values:

C/C++ **enum omp_alloctrait_key_t**
For **integer omp_alloctrait_key_kind**

omp_atk_X where *X* may be one of sync_hint, alignment, access, pool_size, fallback, fb_data, pinned, partition

C/C++ **enum omp_alloctrait_value_t**
For **integer omp_alloctrait_val_kind**

omp_atv_X where *X* may be one of false, true, default, contended, uncontended, serialized, private, all, thread, pteam, cgroup, default_mem_fb, null_fb, abort_fb, allocator_fb, environment, nearest, blocked, interleaved

omp_init_allocator [18.13.2] [3.13.2]

Initializes allocator and associates it with a memory space.

C/C++
```
omp_allocator_handle_t omp_init_allocator (
    omp_memspace_handle_t memspace,
    int ntraits, const omp_alloctrait_t traits[]);
```

Fortran
```
integer (kind=omp_allocator_handle_kind) function &
        omp_init_allocator (memspace, ntraits, traits)
integer (kind=omp_memspace_handle_kind), &
    intent (in) :: memspace
integer, intent (in) :: ntraits
type (omp_alloctrait), intent (in) :: traits (*)
```

omp_destroy_allocator [18.13.3] [3.13.3]

Releases all resources used by the allocator handle.

C/C++
```
void omp_destroy_allocator (
    omp_allocator_handle_t allocator);
```

Fortran
```
subroutine omp_destroy_allocator (allocator)
integer (kind=omp_allocator_handle_kind), &
    intent (in) :: allocator
```

omp_set_default_allocator [18.13.4] [3.13.4]

Sets the default memory allocator to be used by allocation calls, **allocate** directives, and **allocate** clauses that do not specify an allocator.

C/C++
```
void omp_set_default_allocator (
    omp_allocator_handle_t allocator);
```

Fortran
```
subroutine omp_set_default_allocator (allocator)
integer (kind=omp_allocator_handle_kind), &
    intent (in) :: allocator
```

omp_get_default_allocator [18.13.5] [3.13.5]

Returns the memory allocator to be used by allocation calls, **allocate** directives, and **allocate** clauses that do not specify an allocator.

C/C++
```
omp_allocator_handle_t
    omp_get_default_allocator (void);
```

Fortran
```
integer (kind=omp_allocator_handle_kind) &
    function omp_get_default_allocator ()
```

omp_alloc and omp_aligned_alloc [18.13.6] [3.13.6]

Request a memory allocation from a memory allocator.

C
```
void *omp_alloc (size_t size,
    omp_allocator_handle_t allocator);
void *omp_aligned_alloc (size_t alignment,
    size_t size, omp_allocator_handle_t allocator);
```

C++
```
void *omp_alloc (size_t size,
    omp_allocator_handle_t allocator
      = omp_null_allocator);
void *omp_aligned_alloc (size_t size,
    size_t alignment,
    omp_allocator_handle_t allocator
      = omp_null_allocator);
```

Fortran
```
type(c_ptr) function omp_alloc (size, allocator) bind(c)
use, intrinsic :: iso_c_binding, only : c_ptr, c_size_t
integer(c_size_t), value :: size
integer(omp_allocator_handle_kind), value :: allocator
type(c_ptr) function omp_aligned_alloc ( &
    alignment, size, allocator) bind(c)
use, intrinsic :: iso_c_binding, only : c_ptr, c_size_t
integer(c_size_t), value :: alignment, size
integer(omp_allocator_handle_kind), value :: allocator
```

omp_free [18.13.7] [3.13.7]

Deallocates previously allocated memory.

C
```
void omp_free (void *ptr,
    omp_allocator_handle_t allocator);
```

C++
```
void omp_free (void *ptr,
    omp_allocator_handle_t allocator
      = omp_null_allocator);
```

Fortran
```
subroutine omp_free (ptr, allocator) bind(c)
use, intrinsic :: iso_c_binding, only : c_ptr
type(c_ptr), value :: ptr
integer(omp_allocator_handle_kind), value :: allocator
```

omp_calloc and omp_aligned_calloc [18.13.8] [3.13.8]

Request a zero-initialized memory allocation from a memory allocator.

C
```
void *omp_calloc (size_t nmemb, size_t size,
    omp_allocator_handle_t allocator);
void *omp_aligned_calloc (size_t alignment,
    size_t nmemb, size_t size,
    omp_allocator_handle_t allocator);
```

C++
```
void *omp_calloc (size_t nmemb, size_t size,
    omp_allocator_handle_t allocator
      = omp_null_allocator);
void *omp_aligned_calloc (size_t alignment,
    size_t nmemb, size_t size,
    omp_allocator_handle_t allocator
      = omp_null_allocator);
```

Fortran
```
type(c_ptr) function omp_calloc (nmemb, size, &
    allocator) bind(c)
use, intrinsic :: iso_c_binding, only : c_ptr, c_size_t
integer(c_size_t), value :: nmemb, size
integer(omp_allocator_handle_kind), value :: allocator
type(c_ptr) function omp_aligned_calloc ( &
    alignment, nmemb, size, allocator) bind(c)
use, intrinsic :: iso_c_binding, only : c_ptr, c_size_t
integer(c_size_t), value :: alignment, nmemb, size
integer(omp_allocator_handle_kind), value :: allocator
```

Continued

Runtime Library Routines (continued)

omp_realloc [18.13.9] [3.13.9]

Reallocates the given area of memory originally allocated by *free_allocator* using *allocator*, moving and resizing if necessary.

C	**void *omp_realloc (void ***_ptr_**, size_t** _size_**,** **omp_allocator_handle_t** _allocator_**,** **omp_allocator_handle_t** _free_allocator_**);**
C++	**void *omp_realloc (void ***_ptr_**, size_t** _size_**,** **omp_allocator_handle_t** _allocator_ **= omp_null_allocator,** **omp_allocator_handle_t** _free_allocator_ **= omp_null_allocator);**
Fortran	**type(c_ptr) function omp_realloc (&** _ptr_, _size_, _allocator_, _free_allocator_**) bind(c)** **use, intrinsic :: iso_c_binding, only : c_ptr, c_size_t** **type(c_ptr), value ::** _ptr_ **integer(c_size_t), value ::** _size_ **integer(omp_allocator_handle_kind), value :: &** _allocator_, _free_allocator_

Tool control routine

omp_control_tool [18.14] [3.14]

Enables a program to pass commands to an active tool.

C/C++	**int omp_control_tool (int** _command_**, int** _modifier_**,** **void ***_arg_**);**
Fortran	**integer function omp_control_tool (**_command_**, &** _modifier_**)** **integer (kind=omp_control_tool_kind)** _command_ **integer** _modifier_

command:

omp_control_tool_start
Start or restart monitoring if it is off. If monitoring is already on, this command is idempotent. If monitoring has already been turned off permanently, this command will have no effect.

omp_control_tool_pause
Temporarily turn monitoring off. If monitoring is already off, it is idempotent.

omp_control_tool_flush
Flush any data buffered by a tool. This command may be applied whether monitoring is on or off.

omp_control_tool_end
Turn monitoring off permanently; the tool finalizes itself and flushes all output.

Environment display routine

omp_display_env [18.15] [3.15]

Displays the OpenMP version number and the values of ICVs associated with environment variables.

C/C++	**void omp_display_env (int** _verbose_**);**
Fortran	**subroutine omp_display_env (**_verbose_**)** **logical, intent(in) ::** _verbose_

Environment Variables

Environment variable names are upper case. The values assigned to them are case insensitive and may have leading and trailing white space.

OMP_AFFINITY_FORMAT *format* [21.2.5] [6.14]

Sets the initial value of the *affinity-format-var* ICV defining the format when displaying OpenMP thread affinity information. The *format* is a character string that may contain as substrings one or more field specifiers, in addition to other characters. The value is case-sensitive, and leading and trailing whitespace is significant. The format of each field specifier is: %[[[0].]*size*]*type*, where the field type may be either the short or long names listed below [Table 21.2 6.2].

t	team_num	n	thread_num
T	num_teams	N	num_threads
L	nesting_level	a	ancestor_tnum
P	process_id	A	thread_affinity
H	host	i	native_thread_id

OMP_ALLOCATOR [21.5.1] [6.22]

OpenMP memory allocators can be used to make allocation requests. This environment variable sets the initial value of *def-allocator-var* ICV that specifies the default allocator for allocation calls, directives, and clauses that do not specify an allocator. The value is a predefined allocator or a predefined memory space optionally followed by one or more allocator traits.

- Predefined memory spaces are listed in Table 6.1 2.8
- Allocator traits are listed in Table 6.2 2.9
- Predefined allocators are listed in Table 6.3 2.10

Examples

```
setenv OMP_ALLOCATOR omp_high_bw_mem_alloc

setenv OMP_ALLOCATOR \
    omp_large_cap_mem_space : alignment=16, \
    pinned=true

setenv OMP_ALLOCATOR \
    omp_high_bw_mem_space : pool_size=1048576, \
    fallback=allocator_fb, fb_data=omp_low_lat_mem_alloc
```

Memory space names [Table 6.1 2.8]

omp_default_mem_space	**omp_high_bw_mem_space**
omp_large_cap_mem_space	**omp_low_lat_mem_space**
omp_const_mem_space	

Allocator traits & allowed values [Table 6.2 2.9]

sync_hint	**contended, uncontended, serialized, private**
alignment	**1 byte**; Positive integer value that is a power of 2
access	**all, cgroup, pteam, thread**
pool_size	Positive integer value (default is impl. defined)
fallback	**default_mem_fb, null_fb, abort_fb, allocator_fb**
fb_data	An allocator handle (No default)
pinned	**true, false**
partition	**environment, nearest, blocked, interleaved**

Predefined allocators, memory space, and trait values [Table 6.3 2.10]

omp_default_mem_alloc	**omp_default_mem_space** **fallback:null_fb**
omp_large_cap_mem_alloc	**omp_large_cap_mem_space** (none)
omp_const_mem_alloc	**omp_const_mem_space** (none)
omp_high_bw_mem_alloc	**omp_high_bw_mem_space** (none)
omp_low_lat_mem_alloc	**omp_low_lat_mem_space** (none)
omp_cgroup_mem_alloc	Implementation defined **access:cgroup**
omp_pteam_mem_alloc	Implementation defined **access:pteam**
omp_thread_mem_alloc	Implementation defined **access:thread**

OMP_CANCELLATION [21.2.6] [6.11]

Sets the initial value of the *cancel-var* ICV. The value must be **true** or **false**. If **true**, the effects of the **cancel** construct and of cancellation points are enabled and cancellation is activated.

OMP_DEBUG [21.4.1] [6.21]

Sets the *debug-var* ICV. The value must be **enabled** or **disabled**. If **enabled**, the OpenMP implementation will collect additional runtime information to be provided to a third-party tool. If **disabled**, only reduced functionality might be available in the debugger.

OMP_DEFAULT_DEVICE *device* [21.2.7] [6.15]

Sets the initial value of the *default-device-var* ICV that controls the default device number to use in device constructs.

OMP_DISPLAY_AFFINITY *var* [21.2.4] [6.13]

Instructs the runtime to display formatted affinity information for all OpenMP threads in the parallel region. The information is displayed upon entering the first parallel region and when there is any change in the information accessible by the format specifiers listed in the table for **OMP_AFFINITY_FORMAT**. If there is a change of affinity of any thread in a parallel region, thread affinity information for all threads in that region will be displayed. *var* may be **true** or **false**.

OMP_DISPLAY_ENV *var* [21.7] [6.12]

If *var* is **true**, instructs the runtime to display the OpenMP version number and the value of the ICVs associated with the environment variables as *name=value* pairs. If *var* is **verbose**, the runtime may also display vendor-specific variables. If *var* is **false**, no information is displayed.

OMP_DYNAMIC *var* [21.1.1] [6.3]

Sets the initial value of the *dyn-var* ICV. *var* may be **true** or **false**. If **true**, the implementation may dynamically adjust the number of threads to use for executing **parallel** regions.

OMP_MAX_ACTIVE_LEVELS *levels* [21.1.4] [6.8]

Sets the initial value of the *max-active-levels-var* ICV that controls the maximum number of nested active **parallel** regions.

OMP_MAX_TASK_PRIORITY *level* [21.2.9] [6.16]

Sets the initial value of the *max-task-priority-var* ICV that controls the use of task priorities.

OMP_NUM_TEAMS [21.6.1] [6.23]

Sets the maximum number of teams created by a **teams** construct by setting the *nteams-var* ICV.

OMP_NUM_THREADS *list* [21.1.2] [6.2]

Sets the initial value of the *nthreads-var* ICV for the number of threads to use for **parallel** regions.

OMP_PLACES *places* [21.1.6] [6.5]

Sets the initial value of the *place-partition-var* ICV that defines the OpenMP places available to the execution environment. *places* is an abstract name (**threads**, **cores**, **sockets**, **ll_caches**, **numa_domains**) or an ordered list of places where each place of brace-delimited numbers is an unordered set of processors on a device.

Environment Variables

OMP_PROC_BIND *policy* [21.1.7] [6.4]
Sets the initial value of the global *bind-var* ICV, setting the thread affinity policy to use for **parallel** regions at the corresponding nested level. *policy* can have the values **true**, **false**, or a comma-separated list of **primary**, **close**, or **spread** in quotes.

OMP_SCHEDULE *[modifier:]kind[, chunk]* [21.2.1] [6.1]
Sets the *run-sched-var* ICV for the runtime schedule kind and chunk size. *modifier* is one of **monotonic** or **nonmonotonic**; *kind* is one of **static**, **dynamic**, **guided**, or **auto**.

OMP_STACKSIZE *size[* **B** | **K** | **M** | **G** *]* [21.2.2] [6.6]
Sets the *stacksize-var* ICV that specifies the size of the stack for threads created by the OpenMP implementation. *size* is a positive integer that specifies stack size. **B** is bytes, **K** is kilobytes, **M** is megabytes, and **G** is gigabytes. If unit is not specified, *size* is in units of **K**.

OMP_TARGET_OFFLOAD [21.2.8] [6.17]
Sets the initial value of the *target-offload-var* ICV. The value must be one of **mandatory**, **disabled**, or **default**.

OMP_TEAMS_THREAD_LIMIT [21.6.2] [6.24]
Sets the maximum number of OpenMP threads to use in each contention group created by a **teams** construct by setting the *teams-thread-limit-var* ICV.

OMP_THREAD_LIMIT *limit* [21.1.3] [6.10]
Sets the maximum number of OpenMP threads to use in a contention group by setting the *thread-limit-var* ICV.

OMP_TOOL (**enabled**|**disabled**) [21.3.1] [6.18]
Sets the *tool-var* ICV. If disabled, no first-party tool will be activated. If enabled the OpenMP implementation will try to find and activate a first-party tool.

OMP_TOOL_LIBRARIES *library-list* [21.3.2] [6.19]
Sets the *tool-libraries-var* ICV to a list of tool libraries that will be considered for use on a device where an OpenMP implementation is being initialized. *library-list* is a space-separated list of dynamically-linked libraries, each specified by an absolute path.

OMP_TOOL_VERBOSE_INIT [21.3.3] [6.20]
Sets the *tool-verbose-init-var* ICV, which controls whether an OpenMP implementation will verbosely log the registration of a tool. The value must be a filename or one of **disabled**, **stdout**, or **stderr**.

OMP_WAIT_POLICY *policy* [21.2.3] [6.7]
Sets the *wait-policy-var* ICV that provides a hint to an OpenMP implementation about the desired behavior of waiting threads. Valid values for *policy* are **active** (waiting threads consume processor cycles while waiting) and **passive**. Default is implementation defined.

Internal Control Variables (ICV) Values

Host and target device ICVs are initialized before OpenMP API constructs or routines execute. After initial values are assigned, the values of environment variables set by the user are read and the associated ICVs for host and target devices are modified accordingly. Certain environment variables may be extended with device-specific environment variables with the following syntax: <*ENV_VAR*>**_DEV**[_<*device_num*>]. Device-specific environment variables must not correspond to environment variables that initialize ICVs with the global scope.

Table of ICV Initial Values, Ways to Modify and to Retrieve ICV Values, and Scope [Tables 2.1-3] [2.1-3]

ICV	Environment variable	Initial value	Ways to modify value	Ways to retrieve value	Scope	Env. Var. Ref.
active-levels-var	(none)	*zero*	(none)	**omp_get_active_level()**	Data env.	---
affinity-format-var	**OMP_AFFINITY_FORMAT**	Implementation defined.	**omp_set_affinity_format()**	**omp_get_affinity_format()**	Device	[21.2.5] [6.14]
bind-var	**OMP_PROC_BIND**	Implementation defined.	(none)	**omp_get_proc_bind()**	Data env.	[21.1.7] [6.4]
cancel-var	**OMP_CANCELLATION**	*false*	(none)	**omp_get_cancellation()**	Global	[21.2.6] [6.11]
debug-var	**OMP_DEBUG**	*disabled*	(none)	(none)	Global	[21.4.1] [6.21]
def-allocator-var	**OMP_ALLOCATOR**	Implementation defined.	**omp_set_default_allocator()**	**omp_get_default_allocator()**	Impl. Task	[21.5.1] [6.22]
def-sched-var	(none)	Implementation defined.	(none)	(none)	Device	---
default-device-var	**OMP_DEFAULT_DEVICE**	Implementation defined.	**omp_set_default_device()**	**omp_get_default_device()**	Data env.	[21.2.7] [6.15]
display-affinity-var	**OMP_DISPLAY_AFFINITY**	*false*	(none)	(none)	Global	[21.2.4] [6.13]
dyn-var	**OMP_DYNAMIC**	Implementation-defined if the implementation supports dynamic adjustment of the number of threads; otherwise, the initial value is *false*.	**omp_set_dynamic()**	**omp_get_dynamic()**	Data env.	[21.1.1] [6.3]
explicit-task-var	(none)	*false*		**omp_in_explicit_task()**		
final-task-var	(none)	*false*	(none)	**omp_in_final()**	Data env.	---
levels-var	(none)	*zero*	(none)	**omp_get_level()**	Data env.	---
max-active-levels-var	**OMP_MAX_ACTIVE_LEVELS**, **OMP_NUM_THREADS**, **OMP_PROC_BIND**	Implementation defined.	**omp_set_max_active_levels()**	**omp_get_max_active_levels()**	Device Data env.	[21.1.4] [6.8] [21.1.2] [6.9] [21.1.7] [6.2]
max-task-priority-var	**OMP_MAX_TASK_PRIORITY**	*zero*	(none)	**omp_get_max_task_priority()**	Global	[21.2.9] [6.16]
nteams-var	**OMP_NUM_TEAMS**	*zero*	**omp_set_num_teams()**	**omp_get_max_teams()**	Device	[21.6.1] [6.23]
nthreads-var	**OMP_NUM_THREADS**	Implementation defined.	**omp_set_num_threads()**	**omp_get_max_threads()**	Data env.	[21.1.2] [6.2]
num-procs-var	(none)	Implementation defined.	(none)	**omp_get_num_procs()**	Device	---
place-partition-var	**OMP_PLACES**	Implementation defined.	(none)	**omp_get_partition_num_places()** **omp_get_partition_place_nums()** **omp_get_place_num_procs()** **omp_get_place_proc_ids()**	Impl. Task	[21.1.6] [6.5]
run-sched-var	**OMP_SCHEDULE**	Implementation defined.	**omp_set_schedule()**	**omp_get_schedule()**	Data env.	[21.2.1] [6.1]
stacksize-var	**OMP_STACKSIZE**	Implementation defined.	(none)	(none)	Device	[21.2.2] [6.6]
target-offload-var	**OMP_TARGET_OFFLOAD**	**DEFAULT**	(none)	(none)	Global	[21.2.8] [6.17]
team-size-var	(none)	*one*	(none)	**omp_get_num_threads()**	Team	---
teams-thread-limit-var	**OMP_TEAMS_THREAD_LIMIT**	*zero*	**omp_set_teams_thread_limit()**	**omp_get_teams_thread_limit()**	Device	[21.6.2] [6.24]
thread-limit-var	**OMP_THREAD_LIMIT**	Implementation defined.	**target** and **teams** constructs	**omp_get_thread_limit()**	Data env.	[21.1.3] [6.10]
thread-num-var	(none)	*zero*	(none)	**omp_get_thread_num()**	Impl. Task	---
tool-libraries-var	**OMP_TOOL_LIBRARIES**	*empty string*	(none)	(none)	Global	[21.3.2] [6.19]
tool-var	**OMP_TOOL**	*enabled*	(none)	(none)	Global	[21.3.1] [6.18]
tool-verbose-init-var	**OMP_TOOL_VERBOSE_INIT**	*disabled*	(none)	(none)	Global	[21.3.3] [6.20]
wait-policy-var	**OMP_WAIT_POLICY**	Implementation defined.	(none)	(none)	Device	[21.2.3] [6.7]

Using OpenMP Tools

A tool indicates its interest in using the OMPT interface by providing a non-null pointer to an **ompt_start_tool_result_t** structure to an OpenMP implementation as a return value from the **ompt_start_tool** function.

There are three ways that a tool can provide a definition of **ompt_start_tool** to an OpenMP implementation:

- Statically linking the tool's definition of **ompt_start_tool** into an OpenMP application.
- Introducing a dynamically linked library that includes the tool's definition of **ompt_start_tool** into the application's address space.
- Providing the name of a dynamically linked library appropriate for the architecture and operating system used by the application in the *tool-libraries-var* ICV (via **omp_tool_libraries**).

You can use **omp_tool_verbose_init** to help understand issues with loading or activating tools. This runtime library routine sets the *tool-verbose-init-var* ICV, which controls whether an OpenMP implementation will verbosely log the registration of a tool.

OpenMP API 5.2 Reference Guide Index

Clauses-9

Directives and Constructs-1–8
- allocate, allocators-1
- assume, assumes-2
- atomic-6
- barrier-5
- begin assumes-2
- begin declare target-2
- cancel-6
- Cancellation constructs
  - cancel-6
  - cancellation point-6
- cancellation point-6
- Combined forms-6–8
- critical-5
- Data environment directives-1
  - declare mapper-1
  - declare reduction-1
  - scan-1
  - threadprivate-1
- declare mapper-1
- declare reduction-1
- declare simd-2
- declare target-2
- declare variant-2
- depobj-5
- Device directives and construct-4
  - target-5
  - target data-4
  - target enter data-5
  - target exit data-5
  - target update-5
- dispatch-2
- distribute-4
- do-4
- flush-5
- for-4
- Informational & utility directives-2
  - assume, [begin]assumes-2
  - error-3
  - nothing-3
  - requires-2
- interop-5
- loop-4
- Loop transformation constructs-3
  - tile-3
  - unroll-3
- masked-3
- Memory management directives-1
  - allocate-1
  - allocators-1
  - Memory spaces-1
- metadirective-1
- nothing-3
- ordered-6
- parallel-3
- Parallelism constructs-3
  - masked-3
  - parallel-3
  - simd-3
  - teams-3
- requires-2
- scan-1
- scope-3
- section, sections-4
- simd-3
- single-3
- Synchronization constructs-5
  - atomic-6
  - barrier-5
  - critical-5
  - depobj-5
  - flush-5
  - ordered-6
  - taskgroup-5
  - taskwait-5
- target-5
- target data-4
- target enter data-5
- target exit data-5
- target update-5
- task-4
- taskgroup-5
- Tasking constructs-4
  - task-4
  - taskloop-4
  - taskyield-4
- taskloop-4
- taskwait-5
- taskyield-4
- teams-3
- threadprivate-1
- tile-3
- unroll-3
- Variant directives-1
  - [begin]declare target-2
  - declare simd-2
  - declare variant-2
  - dispatch-2
  - metadirective-1
- Work-distribution constructs-3
  - distribute-4
  - do-4
  - for-4
  - loop-4
  - scope-3
  - section, sections-4
  - single-3
  - workshare-3
- workshare-3

Environment Variables-14–15

Internal Control Variable (ICV) Values-15

Runtime Library Routines-10–14
- Device information routines-11
- Device memory routines-11–12
- Environment display routine-14
- Event routine-13
- Interoperability routines-13
- Lock routines-12–13
- Memory management routines-13
- Resource relinquishing routines-11
- Tasking routines-11
- Teams region routines-11
- Thread affinity routines-10–11
- Thread team routines-10
- Timing routines-13
- Tool control routine-14