OpenMP 6.0 API Syntax Reference Guide

The OpenMP® API is a scalable model that gives programmers a simple and flexible interface for developing portable parallel applications in C/C++ and Fortran. OpenMP is suitable for a 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 | *For*: Fortran content | **[n.n.n]**: 6.0 spec section | **[n.n.n]**: 5.2 spec section | C: Clause info pg. 10 | _: Underscore optional

Getting Started

Navigating this reference guide

OpenMP code examples

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

OpenMP directive format

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

C/C++ C/C++ directives are formed with pragmas alone or pragmas with attributes.

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

See **[5]** **[5]** for details about the directive format.

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, a following structured block. An underscore is required in the **directive_name** unless indicated with _. • 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 is not 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 [7.3] [5.2]

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***)**
For	**!$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 [7.6.14] [5.5.11] _

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***)** *[clause[[,] clause]*
Fortran	**!$omp declare_reduction &** **(***reduction-identifier* **:** *typename-list***)** *[clause[[,] clause]*

reduction-identifier:

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

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

clause:

combiner (*combiner-expr***)**

combiner-expr (REQUIRED)
C/C++ An expression including function calls

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

Use **omp_out** as the result, and **omp_in** and **omp_out** as the two input operands for the reduction operation.

CONTINUES IN NEXT COLUMN >

declare_reduction (continued)

initializer (*initializer-expr***)**

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

declare_induction [7.6.17]

Declares an *induction-identifier* that can used in the **induction** clause.

C/C++	**#pragma omp declare_induction (** *induction-identifier* **:** *type-specifier-list* **)** *clause [,] clause*
For	**!$omp declare_induction (** *induction-identifier* **:** *type-list***)** *clause [,] clause*

clause:

Both the **collector** and **inductor** clauses must be specified.

collector (*collector-expression***)**

collector-expression: *expression*, use **omp_idx** for loop counter and **omp_step** for loop increment.

inductor (*inductor-expression***)**

inductor-expression: **omp_var** = *expression*, use **omp_var** for induction variable and **omp_step** for loop increment.

scan [7.7] [5.6]

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** *init-complete-clause***]** *structured-block-sequence* **#pragma omp scan** *scan-clause* *structured-block-sequence* **}**
Fortran	*structured-block-sequence* **[!$omp scan** *init-complete-clause***]** *structured-block-sequence* **#!$omp scan** *scan-clause* *structured-block-sequence*

clause:

scan-clause

exclusive (*list***)**
inclusive (*list***)**

init-complete-clause

init_complete (*omp-constant-logical-expression***)**

declare_mapper [7.9.10] [5.8.8] _

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]...]*
For	**!$omp declare_mapper (***[mapper-identifier* **:** *]type* **::** *var***) &** *[clause[[,] clause] ...]*

clause:

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

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

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

mapper-identifier:

A base-language identifier or **default** C

type: A valid type in scope

var: A valid base-language identifier

groupprivate [7.13]

Distribute the list items such that each contention group receives its own copy.

C/C++	**#pragma omp groupprivate** *[clause]*
For	**!$omp declare groupprivate** *[clause]*

clause:

device_type (host | nohost | any)

Specify type of device for which a version of the procedure or variable should be made available.

Memory management directives

Memory spaces [8.1] [6.1]

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

Memory space	Storage selection intent
omp_default_mem_space	System default storage
omp_large_cap_mem_space	Storage with large capacity
omp_const_mem_space	Storage best for constant-value variables
omp_high_bw_mem_space	Storage with high bandwidth
omp_low_lat_mem_space	Storage with low latency

Continued

Directives and Constructs (continued)

allocate [8.5] [6.6]

Specifies how a set of variables is allocated.

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

For: **!$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 [8.7] [6.7]

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

Fortran:
!$omp allocators *[clause]*
allocate-stmt
[!$omp end allocators]

clause: **allocate** C

allocate-stmt: A Fortran ALLOCATE statement.

Variant directives

[begin]metadirective [9.4.3–4] [7.4.3–4]

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.

[begin]declare_variant [9.6.4–5] [7.5.4–5]

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 *match-clause*
declaration-definition-seq
#pragma omp end declare_variant

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

clause:

adjust_args (*adjust-op* **:** *argument-list***)**
Specifies how to adjust the arguments of the base function when a specified variant function is selected for replacement.
adjust-op: **need_device_addr**, **need_device_ptr**, **nothing**

CONTINUES IN NEXT COLUMN >

[begin]declare_variant (continued)

append_args (*append-op[[,* *append-op]...]***)**
Specifies additional argument to append to the argument list of the call to the variant function
append-op: **interop (**
*interop-type[[, interop-type]...]***)**

match (*context-selector-specification***)**
Specifies conditions in which variant function is selected for replacement.

match-clause:

match (*context-selector-specification***)**
REQUIRED. Same as above

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.

dispatch [9.7] [7.6]

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***)** C
device (*omp-integer-expression***)** C
Identifies the target device that is associated with a device construct.
interop (*interop-var-list***)**
is_device_ptr (*list***)**
list: device pointers
has_device_addr (*list-items***)**
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 C

declare_simd [9.8] [7.7]

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

For:
!$omp declare_simd *[***(***proc-name***)***]* *[clause[[,]clause] ...]*

clauses:

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
linear (*linear-list***[: ***linear-step***])** C
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 [9.9.1–2] [7.8.1–2]

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)
Specify type of device for which a version of the procedure or variable should be made available.

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.

local (*list-items***)**
Specify that references to *list-items* refer to a local copy of the *list-items* on the device.

- For the second C/C++ form of **declare_target**, at least one clause must be **enter** or **link**.
- For **begin declare_target**, only the clauses **indirect** and **device_type** are permitted.

Informational and utility directives

requires [10.5] [8.2]

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] ...]*

For: **!$omp requires** *clause [[[,] clause] ...]*

clause:

atomic_default_mem_order (seq_cst | acq_rel | relaxed)
Requires that implementation uses the specified memory ordering for atomic constructs that do not specify a **memory-order** clause.

device_safesync
Requires that synchronizing divergent threads in a team can make progress on a non-host device, unless a **safesync** clause is specified.

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.)

self_maps
Requires that all **map** clauses of map-entering directives have an implicit **self** modifier.

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.

Continued

Directives and Constructs (continued)

assume, [begin]assumes [10.6.2–4] [8.3.2–4]

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
Equivalent to union of the **no_openmp_constructs** and **no_openmp_routines** assumptions.

no_openmp_constructs
Indicates that no OpenMP constructs are encountered 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.

nothing [10.7] [8.4]

Indicates explicitly that the intent is to have no effect.

C/C++
```
#pragma omp nothing [clause]
```

For
```
!$omp nothing [clause]
```

clause:

apply (*applied-directives*) C
Can be specified only if the **nothing** directive forms a loop-transforming construct.

error [10.1] [8.5]

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

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

For
```
!$omp error [clause [ [,] clause] ... ]
```

clause:

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

Loop-transforming constructs

fuse [11.3]

Merges multiple loops into a single loop that in each iteration executes an iteration of each input loop.

C/C++
```
#pragma omp fuse clause
    loop-nest-sequence
```

Fortran
```
!$omp fuse clause
    loop-nest-sequence
[!$omp end fuse]
```

clause:

apply (*applied-directives*) C
looprange (*list of loop nests*)

interchange [11.4]

Changes the nesting order of loops in a loop nest.

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

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

clause:

apply (*applied-directives*) C
permutation (*permutation-list*)

reverse [11.5]

Executes loop iterations in reverse order.

C/C++
```
#pragma omp reverse [clause]
    loop-nest
```

Fortran
```
!$omp reverse [clause]
    loop-nest
[!$omp end reverse]
```

clause: **apply** (*applied-directives*) C

split [11.6]

Splits a loop into multiple loops that each execute a subset of consecutive iterations.

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

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

clause:

apply (*applied-directives*) C
counts (*count-list*)

stripe [11.7]

Interleaves execution of loop iterations with respect to the partitions that result from the **sizes** clause.

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

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

clause:

apply (*applied-directives*) C
sizes (*size-list*)

tile [11.8] [9.1]

Tiles one or more loops.

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

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

clause:

apply (*applied-directives*) C
sizes (*size-list*)

unroll [11.9] [9.2]

Fully or partially unrolls a loop.

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

Fortran
```
!$omp unroll [clause]
    loop-nest
[!$omp end unroll]
```

clause:

apply (*applied-directives*) C
full
partial *[*(*unroll-factor*)*]*

Parallelism constructs

parallel [12.1] [10.1]

Creates a team of OpenMP threads that executes 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 C
copyin (*list*)
default (*data-sharing-attribute*) C
firstprivate (*list*) C
if (*[*parallel **:** *] omp-logical-expression*) C
message
num_threads (*nthreads*)
Specifies the number of threads to execute.
private (*list*) C
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 C
safesync (*width*)
severity (**fatal** | **warning**)
shared (*list*) C

teams [12.2] [10.2]

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

Continued

Directives and Constructs (continued)

simd [12.4] [10.4]

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*) C
if ([**simd** :] *omp-logical-expression*) C
induction C
lastprivate ([*lastprivate-modifier* :] *list*) C
linear (*list*[: *linear-step*]) C
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**) C
order-modifier: **reproducible** or **unconstrained**
private (*list*) C
reduction C
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 [12.5] [10.5]

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 the structured block.

Work-distribution constructs

single [13.1] [11.1]

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 C
copyprivate (*list*)
firstprivate (*list*) C
nowait C
private (*list*) C

end-clause:

copyprivate (*list*)
nowait C

scope [13.2] [11.2]

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 C
firstprivate (*list*) C
nowait C
private (*list*) C
reduction C

section and sections [13.3–13.3.1] [11.3–11.3.1]

A non-iterative worksharing construct that contains a set of structured block sequences 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 C
firstprivate (*list*) C
lastprivate ([*lastprivate-modifier*:] *list*) C
nowait C
private (*list*) C
reduction C

workshare [13.4] [11.4]

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

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

clause:

nowait C

workdistribute [13.5]

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

Fortran:
```
!$omp workdistribute
    loosely structured-block
!$omp end distribute
- or -
!$omp workdistribute
    strictly structured-block
[!$omp end distribute]
```

do and for [13.6.1–2] [11.5.1–2]

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

Values for **schedule** *modifier:*

monotonic: Each thread executes its assigned chunks in increasing logical iteration order. Clauses **schedule** (static) or **order** imply monotonic.

nonmonotonic: Chunks are assigned to threads in any order. 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 the new *chunk_size* for all chunks except the first and last chunks is ⌈*chunk_size*/*simd_width*⌉ * *simd_width* (*simd_width*: implementation-defined value).

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.

distribute [13.7] [11.6]

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

Continued

Directives and Constructs (continued)

loop [13.8] [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*) C
- **lastprivate** (*[lastprivate-modifier :] list*) C
- **order** (*[order-modifier :]* **concurrent**) C
  order-modifier: **reproducible** or **unconstrained**
- **private** (*list*) C
- **reduction** C

Tasking constructs

task [14.1] [12.5]

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*) C
- **allocate** C
- **default** (*data-sharing-attribute*) C
- **depend** (*[depend-modifier,] dependence-type: locator-list*) C
- **detach** (*event-handle*)
  Task does not complete until given event is fulfilled. (Also see **omp_fulfill_event**)
  event-handle:
  C/C++ type **omp_event_handle_t**
  For kind **omp_event_handle_kind**
- **final** (*omp-logical-expression*)
  The generated task will be a final task if the expression evaluates to true.
- **firstprivate** (*list*) C
- **if** (*[task :] omp-logical-expression*) C
- **in_reduction** (*reduction-identifier : list*) C
- **mergeable**
- **priority** (*priority-value*) C
  Hint to the runtime. Sets max priority value.
- **private** (*list*) C
- **replayable** C
- **shared** (*list*) C
- **threadset** (**omp_pool** | **omp_team**) C
- **transparent** (*impex-type*)
  impex-type can be **omp_import**, **omp_export**, **omp_impex**, **omp_not_impex**.
- **untied**
  Task is untied: any thread in the binding thread set can resume the task region after suspension.

taskloop [14.2] [12.6]

Specifies that the iterations of one or more affected 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** C
- **collapse** (*n*) C

CONTINUES IN NEXT COLUMN >

taskloop (continued)

- **default** (*data-sharing-attribute*) C
- **final** (*omp-logical-expression*)
  The generated tasks will be final tasks if the expression evaluates to true.
- **firstprivate** (*list*) C
- **grainsize** (*[strict :] grain-size*)
  Causes number of logical loop iterations assigned to each created task to be ≥ 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*) C
- **in_reduction** (*reduction-identifier : list*) C
- **induction** C
- **lastprivate** (*[lastprivate-modifier :] list*) C
- **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*) C
- **private** (*list*) C
- **reduction** C
- **replayable** C
- **shared** (*list*) C
- **threadset** C
- **untied**
  Task is untied: any thread in the binding thread set can resume the task region after suspension.

task_iteration [14.2.3]

Controls the per-iteration task-execution attributes of the generated tasks of its associated taskloop construct. Enables the application of task dependences and task affinity to tasks generated by the taskloop construct.

C/C++:
```
#pragma omp task_iteration clause [ [ [,]clause] ... ]
```
For:
```
!$omp task_iteration clause [ [ [,]clause] ... ]
```

clause:

- **affinity** C
- **depend** C
- **if** (*[task_iteration :] omp-logical-expression*) C

taskyield [14.12] [12.7]

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

C/C++:
```
#pragma omp taskyield
```
For:
```
!$omp taskyield
```

taskgraph [14.3]

Executes the structured block and records the sequence and dependences of created tasks for later replaying.

C/C++:
```
#pragma omp taskgraph [clause[ [,]clause] ... ]
```
For:
```
!$omp taskgraph [clause[ [,]clause] ... ]
```

clause:

- **graph_id**(*graph_id*)
  Numeric identification to distinguish graphs for recording and replay.
- **graph_reset**(*omp-logical-condition*)
  Reset recorded graph with *graph-id* when condition evaluates to true
- **if** (*[taskgraph :] omp-logical-expression*) C
- **nogroup**
  Prevents creation of implicit taskgroup region.

Device directives and constructs

target_data [15.7] [13.5]

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:

- **affinity** C
- **allocate**
- **default** (*data-sharing-attribute*) C
- **depend** C
- **detach**
- **device** (*omp-integer-expression*) C
- **firstprivate** (*list*) C
- **if** (*[target_data :] omp-logical-expression*) C
- **in_reduction**
- **map** (*[[map-modifier,] [map-modifier, ...]] map-type :] list*) C
- **mergeable**
- **nogroup**
  Prevents creation of implicit **taskgroup** region.
- **nowait** C
- **private** (*list*) C
- **priority** (*priority-value*) C
- **shared** (*list*) C
- **transparent** (*impex-type*)
  impex-type can be **omp_import**, **omp_export**, **omp_impex**, **omp_not_impex**.
- **use_device_ptr** (*list*)
- **use_device_addr** (*list*)

target_enter_data [15.5] [13.6]

Maps variables to a device data environment.

C/C++:
```
#pragma omp target_enter_data [clause[ [,]clause] ... ]
```
For:
```
!$omp target_enter_data [clause[ [,]clause] ... ]
```

clause:

- **depend** (*[depend-modifier,] dependence-type : locator-list*) C
- **device** (*omp-integer-expression*) C
- **if** (*[target_data :] omp-logical-expression*) C
- **map** (*[[map-modifier,] [map-modifier, ...]] map-type :] list*) C
- **nowait** C
- **priority** C
- **replayable** C

target_exit_data [15.6] [13.7]

Unmaps variables from a device data environment.

C/C++:
```
#pragma omp target_exit_data [clause[ [,]clause] ... ]
```
For:
```
!$omp target_exit_data [clause[ [,]clause] ... ]
```

clause: Any clause used for **target_enter_data**.

target [15.8] [13.8]

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]
```

CONTINUES ON NEXT PAGE >

Continued

Directives and Constructs (continued)

target (continued)

clause:

allocate C
default C
defaultmap (*implicit-behavior* [: *variable-category*])
implicit-behavior: **storage**, **default**, **firstprivate**, **from**, **none**, **present**, **to**, **tofrom**
variable-category: **aggregate**, **all**, **pointer**, **scalar**, *For* **allocatable**
depend (*[depend-modifier,] dependence-type* : *locator-list*) C
device(*[device-modifier:] omp-integer-expression*) C
device-modifier: **ancestor, device_num**
device_type (**any** | **host** | **nohost**)
Specify the type of the device the target region should be made available for.
firstprivate (*list*) C
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*) C
in_reduction (*reduction-identifier* : *list*) C
is_device_ptr(*list*)
Indicates *list* items are device pointers.
map (*[[map-modifier, [map-modifier, ...]] map-type :] list*) C
nowait C
private (*list*) C
priority C
replayable C
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 [15.9] [13.9]

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] ...]*
For: **!$omp target_update** *clause[[[,]clause] ...]*

clause:

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

Interoperability construct

interop [16.1] [14.1]

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

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

clause:

device(*omp-integer-expression*) C
depend (*[depend-modifier,] dependence-type* : *locator-list*) C
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 C
use(*interop-var*)

Synchronization constructs

critical [17.2] [15.2]

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_contended
omp_sync_hint_none
omp_sync_hint_nonspeculative
omp_sync_hint_speculative
omp_sync_hint_uncontended

barrier [17.3] [15.3.1]

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**
For: **!$omp barrier**

taskgroup [17.4] [15.4]

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 C
task_reduction (*reduction-identifier* : *list*)
reduction-identifier: See **reduction** C

taskwait [17.5] [15.5]

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

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

clause:

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

atomic [17.8.5] [15.8.4]

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 comparison performed by a conditional atomic update may spuriously fail, evaluating to not equal even when values are equal.
hint (*hint-expression*)
memscope (**device** | **cgroup** | **all**)
Determines the thread set affected by the atomic operation (**device:** all threads on the current device; **cgroup:** all threads in the current contention group; **all:** all threads in the system).

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*; } }

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

CONTINUES ON NEXT PAGE >

Continued

Directives and Constructs (continued)

atomic (continued)

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

flush [17.8.6] [15.8.5]

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*)*]*

For: **!$omp flush** *[memory-order-clause] [*(*list*)*]*

memory-order-clause:

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

memscope (device | cgroup | all)

Determines the thread set affected by the atomic operation (**device:** all threads on the current device; **cgroup:** all threads in the current contention group; **all:** all threads in the system).

depobj [17.9.3] [15.9.4]

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

C/C++: **#pragma omp depobj (***depend-object***)** *clause*

For: **!$omp depobj (***depend-object***)** *clause*

clause:

destroy (*depend-object***)**

init (*dependence-type***(***list-item***):** *depend-object***)**

Initialize the depend object in *depend-object*.

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

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**

ordered [17.10] [15.10.2]

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.

CONTINUES IN NEXT COLUMN >

ordered (continued)

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 [18.2] [16.1]

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

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

For: **!$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 [18.3] [16.2]

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*

For: **!$omp cancellation_point** *construct-type-clause*

construct-type-clause:

parallel

sections

taskgroup

C/C++ **for**

For **do**

Combined Constructs

The following compound directives are created following the rules defined in Section 19 and Appendix D of the OpenMP API version 6.0 specification. The combined directives shown here are examples of commonly useful compositions of directives and are not a complete list.

distribute parallel do and distribute parallel for

[19] [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**.

distribute parallel do simd and distribute parallel for simd [19] [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**.

distribute simd [19] [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 clause used for **distribute** or **simd**.

Continued

Combined Constructs (continued)

do simd and for simd [19] [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 clause used for **simd**, **for**, or **do**.

masked taskloop [19] [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 [19] [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 do and parallel for [19] [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 do simd and parallel for simd [19] [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 loop [19] [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 masked [19] [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**.

parallel masked taskloop [19] [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**.

parallel masked taskloop simd [19] [2.16.10]

Shortcut to specify 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**.

parallel target [19]

Specifies a **parallel** construct that only contains a **target** construct.

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

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

target loop [19] [2.16.15]

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

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

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

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

target teams [19] [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 [19] [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 [19] [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 [19] [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 [19] [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 [19] [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**.

Continued

Combined Constructs (continued)

taskloop simd [19] [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**.

teams distribute [19] [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 parallel do and teams distribute parallel for [19] [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 [19] [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 distribute simd [19] [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**.

Notes

Clauses

Modifier: iterator [5.2.6] [3.2.6]

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

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

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.

Used in clauses: affinity, depend, from, map, to

Modifier: directive name [5.4]

directive-name-modifier

Each clause can have an optional directive-name-modifier that corresponds to the name of a directive or compound directive that can accept that clause. Examples:

C/C++

#pragma parallel private(parallel: list)
Specifies that the items in list are private to the parallel region (default for this example).

For

$!omp target teams distribute parallel do
if(*target:condition1***) if(***parallel:condition2***)**
Specifies that *condition1* applies to the target constituent directive, while *condition2* applies to the parallel constituent directive.

Used in clauses: All clauses

affinity clause [14.10] [12.5.1]

affinity (*locator-list***)**

Specifies a hint to indicate data affinity of the generated tasks to the data specified in the *locator-list*.

Used in: target_data, task, task_iteration

allocate clause [8.6] [6.6]

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: allocators, distribute, do and for, parallel, scope, sections, single, target, target_data, task, taskgroup, taskloop, teams

apply clause [11.1]

apply (*[loop-modifier* **:** *] applied-directives***)**

After applying a loop-transforming construct, inserts applied-directives in front of the after-transformation loop identified by *loop-modifier*.

Used in: fuse, interchange, nothing, reverse, split, stripe, tile, unroll

collapse clause [4.4.5] [4.4.3]

collapse (*n***)**

A constant positive integer expression that specifies how many loops are affected by the construct.

Used in: distribute, do and for, loop, simd, taskloop

default clause [7.5.1] [5.4.1]

default (*attribute***)**

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.

attribute: **shared**, **firstprivate**, **private**, **none**

Used in: parallel, target, target_data, task, taskloop, teams

depend clause [17.9.5] [15.9.5]

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**: 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**: Generated task will be a dependent task of previously generated sibling tasks referencing at least one list item in an **in**, **out**, **mutexinoutset**, **inout**, or **inoutset** *dependence-type* list.
- **mutexinoutset**: If the storage location of at least one list item is the same as that of a list item 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.
- **inoutset**: If the storage location of at least one list item matches the storage location of a list item 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: dispatch, interop, target, target_enter_data, target_exit_data, target_update, task, task_iteration, taskwait

device clause [15.2] [13.2]

device (*[target-device* **:** *] device-description***)**

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

target-device: **ancestor**, **device_num**

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, interop, target, target_data, target_enter_data, target_exit_data, target_update

firstprivate clause [7.5.4] [5.4.4]

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, do and for, parallel, scope, sections, single, target, target_data, task, taskloop, teams

if clause [5.5] [3.4]

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

The effect of the **if** clause depends on the construct to which it is applied. For compound 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 compound construct then the **if** clause applies to all constructs to which an **if** clause can apply.

Used in: cancel, parallel, sim, target, target_data, target_enter_data, target_exit_data, target_update, task, task_iteration, taskgraph, taskloop, teams

induction clause [7.6.13]

induction (*induction-identifier [, induction-modifier] [, step-modifier] : list***)**

Computes the closed form of an expression for the iterations of a loop.

induction-identifier: Either an *id-expression* or one of the following operators: **+**, ***

induction-modifier: **strict**, **relaxed**
strict: Compute the induction value for each iteration and assign to the original item in list.
relaxed: Implementation may assume that the value of the closed form is computed at the end without assigning to the original item in list in each iteration.

step-modifier: **step(***induction-step***)**
The increment to be applied for each iteration when computing the closed form.

Used in: distribute, do and for, simd, taskloop

in_reduction clause [7.6.12] [6.5.11]

in_reduction (*reduction-identifier* **:** *list***)**

Defines a task to be a participant in a task reduction that is defined by an enclosing region for a matching list item that appears in a **task_reduction** clause or a **reduction** clause with **task** as the *reduction-modifier*.

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: target, target_data, task, taskloop

lastprivate clause [7.5.5] [5.4.5]

lastprivate (*[lastprivate-modifier***:***] list***)**

List items must be loop iteration variables of the affected loops. Same effect as the **private** clause, and after leaving the region, also assigns the values that each variable had at the end of the last iteration of a loop back to the original variable.

lastprivate-modifier: **conditional**
conditional: Preserves the previous value if a variable is not assigned a new value in the last iteration of the loop.

Used in: distribute, do and for, loop, sections, simd, taskloop

linear clause [7.5.6] [5.4.6]

linear (*linear-list***[:** *linear-step***])**
linear (*linear-list* **[:** *linear-modifier* **[**, *linear-modifier***]])**

Declares each *linear-list* item to have a value or address affine with respect to the iteration number.

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, do and for, simd

Continued

Clauses (continued)

map clause [7.9.6] [5.8.3]

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, target, target_data, target_enter_data, target_exit_data

nowait clause [17.6] [15.6]

nowait (*[value]*)

value: **true**, **false**

If no argument is specified, it will default to true. 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, do and for, interop, scope, sections, single, target, target_data, target_enter_data, target_exit_data, target_update, taskwait, workshare

order clause [12.3] [10.3]

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, do and for, loop, simd

priority clause [14.9] [12.4]

priority (*priority-value*)

Specifies, in the *priority-value* argument, a task priority for the construct on which it appears.

Used in: target, target_data, target_enter_data, target_exit_data, target_update, task, taskgraph, taskloop

private clause [7.5.3] [5.4.3]

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, do and for, loop, parallel, scope, sections, simd, single, target, target_data, task, taskloop, teams

reduction clause [7.6.10] [5.5.8]

reduction (*[reduction-modifier* **,***] reduction-identifier* **:** *list*)

Combines the values a variable has in each task or SIMD lane into a single value.

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

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

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

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, loop, parallel, scope, sections, simd, taskloop, teams

replayable clause [14.6]

replayable

Marks the generated task for recording and replay for the **taskgraph** construct.

Used in: target, target_enter_data, target_exit_data, target_update, task, taskloop, taskwait

shared clause [7.5.2] [5.4.2]

shared (*list*)

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

Used in: parallel, target_data, task, taskloop, teams

task_reduction clause [7.6.11, 7.6.7] [5.5.9]

task_reduction (*reduction-identifier* **:** *list*)

A reduction-scoping clause that defines the region in which a reduction is computed by tasks or SIMD lanes via the **in_reduction** clause.

See **reduction** clause for details about *reduction-identifier*.

Used in: taskgroup

threadset clause [14.8]

threadset (*set*)

Specifies the set of threads that may execute tasks that are generated by the construct on which it appears. If the encountering task is a final task, the **threadset** clause is ignored.

set: **omp_team**, **omp_pool**

Used in: task, taskloop

Runtime Library Routines

Parallel region support routines

omp_set_num_threads [21.1] [18.2.1]

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***);**
For	**subroutine omp_set_num_threads (***num_threads***)** **integer** *num_threads*

omp_get_num_threads [21.2] [18.2.2]

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);**
For	**integer function omp_get_num_threads ()**

omp_get_thread_num [21.3] [18.2.4]

Returns the thread number of the calling thread within the current team, ranging from 0 to one less than the total number of threads.

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

omp_get_max_threads [21.4] [18.2.3]

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);**
For	**integer function omp_get_max_threads ()**

omp_get_thread_limit [21.5] [18.2.13]

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

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

omp_in_parallel [21.6] [18.2.5]

Returns true if the *active-levels-var* ICV is greater than zero, meaning the current task is enclosed by an active parallel region; otherwise it returns false.

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

omp_set_dynamic [21.7] [18.2.6]

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***);**
For	**subroutine omp_set_dynamic (***dynamic_threads***)** **logical** *dynamic_threads*

omp_get_dynamic [21.8] [18.2.7]

Returns the value of *dyn-var* ICV. True means dynamic adjustment of the number of threads is enabled for the current task. Use **+** or **|** (C/C++) or **+** (For) to combine kinds with the modifier *omp_sched_monotonic* (see [20.5.1][18.2.11]).

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

Continued

Runtime Library Routines (continued)

omp_set_schedule [21.9] [18.2.11]
Affects the schedule 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 (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_schedule [21.10] [18.2.12]
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 (omp_sched_kind)** *kind* **integer** *chunk_size*

See **omp_set_schedule** for *kind*.

omp_get_supported_active_levels [21.11] [18.2.14]
Returns the number of active levels of parallelism supported.

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

omp_set_max_active_levels [21.12] [18.2.15]
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*);
For	**subroutine omp_set_max_active_levels (***max_levels*) **integer** *max_levels*

omp_get_max_active_levels [21.13] [18.2.16]
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);**
For	**integer function omp_get_max_active_levels ()**

omp_get_level [21.14] [18.2.17]
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);**
For	**integer function omp_get_level ()**

omp_get_ancestor_thread_num [21.15] [18.2.18]
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 [21.16] [18.2.19]
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*);
For	**integer function omp_get_team_size (***level*) **integer** *level*

omp_get_active_level [21.17] [18.2.20]
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);**
For	**integer function omp_get_active_level ()**

Teams region routines

omp_get_num_teams [22.1] [18.4.1]
Returns the number of initial teams in the current **teams** region.

C/C++	**int omp_get_num_teams (void);**
For	**integer function omp_get_num_teams ()**

omp_set_num_teams [22.2] [18.4.3]
Sets the value of the *nteams-var* ICV of the current device, affecting the number of teams 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*);
For	**subroutine omp_set_num_teams(***num_teams*) **integer** *num_teams*

omp_get_team_num [22.3] [18.4.2]
Returns the value of the *team-num-var* ICV, which is the team number of the current team and is an integer between 0 and one less than the number of teams in the currents teams region.

C/C++	**int omp_get_team_num (void);**
For	**integer function omp_get_team_num ()**

omp_get_max_teams [22.4] [18.4.4]
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);**
For	**integer function omp_get_max_teams()**

omp_get_teams_thread_limit [22.5] [18.4.6]
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);**
For	**integer function omp_get_teams_thread_limit ()**

omp_set_teams_thread_limit [22.6] [18.4.5]
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*

Tasking support routines

omp_get_max_task_priority [23.1.1] [18.5.1]
Returns the maximum value that can be specified in the **priority** clause.

C/C++	**int omp_get_max_task_priority (void);**
For	**integer function omp_get_max_task_priority ()**

omp_in_explicit_task [23.1.2]
Returns true if the encountering task is an explicit task region.

C/C++	**int omp_in_explicit_task (void);**
For	**logical function omp_in_explicit_task ()**

omp_in_final [23.1.3] [18.5.3]
Returns true if the routine is executed in a final task region; otherwise, it returns false.

C/C++	**int omp_in_final (void);**
For	**logical function omp_in_final ()**

omp_is_free_agent [23.1.4]
Returns true if a free-agent thread is executing the enclosing task region at the time the routine is called.

C/C++	**int omp_is_free_agent (void);**
For	**logical function omp_is_free_agent ()**

omp_ancestor_is_free_agent [23.1.5]
Returns true if the ancestor thread of the encountering thread is a free-agent thread.

C/C++	**int omp_ancestor_is_free_agent (int** *level*);
For	**logical function omp_ancestor_is_free_agent (***level*) **integer** *level*

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 [23.2.1] [18.11.1]
Fulfills and destroys the event associated with the *event* argument.

C/C++	**void omp_fulfill_event (omp_event_handle_t** *event*);
For	**subroutine omp_fulfill_event (***event*) **integer (omp_event_handle_kind)** *event*

Device information routines

omp_set_default_device [24.1] [18.7.2]
Assigns the value of the *default-device-var* ICV, which determines default target device.

C/C++	**void omp_set_default_device (int** *device_num*);
For	**subroutine omp_set_default_device (***device_num*) **integer** *device_num*

omp_get_default_device [24.2] [18.7.3]
Returns the value of the *default-device-var* ICV, which is the device number of the default target device.

C/C++	**int omp_get_default_device (void);**
For	**integer function omp_get_default_device ()**

Continued

Runtime Library Routines (continued)

omp_get_num_devices [24.3] [18.7.4]
Returns the number of non-host devices available for offloading code or data.

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

omp_get_device_num [24.4] [18.7.5]
Returns the device number of the device on which the calling thread is executing.

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

omp_get_num_procs [24.5] [18.7.1]
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);
```
For:
```
integer function omp_get_num_procs ()
```

omp_get_max_progress_width [24.6]
Returns the maximum size, in terms of hardware threads, of progress units on the device specified by *device_num*.

C/C++:
```
int omp_get_max_progress_width (int device_num);
```
Fortran:
```
integer function omp_get_max_progress_width &
    (device_num)
integer device_num
```

omp_get_device_from_uid [24.7]
Returns the device number that is associated with the unique string in *uid*, or **omp_invalid_device** if the device does not exist.

C/C++:
```
int omp_get_device_from_uid (
    const char *uid);
```
For:
```
integer function omp_get_device_from_uid (uid)
character (len=*), intent(in) :: uid
```

omp_get_uid_from_device [24.8]
Returns the unique string that identifies the device *device_num*, or NULL if *device_num* has the value **omp_invalid_device**

C/C++:
```
const char *omp_get_uid_from_device (
    int device_num);
```
Fortran:
```
character(:) function omp_get_uid_from_device &
    (device_num)
pointer :: omp_get_uid_from_device
integer, intent(in) :: device_num
```

omp_is_initial_device [24.9] [18.7.6]
Returns true if the current task is executing on the host device; otherwise, it returns false.

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

omp_get_initial_device [24.10] [18.7.7]
Returns the device number of the host device.

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

omp_get_device_num_teams [24.11]
Returns the number of teams that will be requested for a teams region on device *device_num* if the **num_teams** clause is not specified.

C/C++:
```
int omp_get_device_num_teams (
    int device_num);
```
Fortran:
```
integer function omp_get_device_num_teams &
    (device_num)
integer device_num
```

omp_set_device_num_teams [24.12]
Sets the number of teams that will be requested for a teams region on device *device_num* if the **num_teams** clause is not specified.

C/C++:
```
void omp_set_device_num_teams (
    int num_teams, int device_num);
```
Fortran:
```
subroutine omp_set_device_num_teams &
    (num_teams, device_num)
integer num_teams, device_num
```

omp_get_device_teams_thread_limit [24.13]
Returns max number of threads available to execute tasks in each contention group that a **teams** construct creates.

C/C++:
```
int omp_get_device_teams_thread_limit (
    int device_num);
```
Fortran:
```
integer function omp_get_device_teams_thread_limit &
    (device_num)
integer device_num
```

omp_set_device_teams_thread_limit [24.14]
Defines max number of threads available to execute tasks in each contention group that a **teams** construct creates.

C/C++:
```
void omp_set_device_teams_thread_limit (
    int thread_limit, int device_num);
```
Fortran:
```
subroutine function omp_set_device_teams_ &
    thread_limit (thread_limit, device_num)
integer thread_limit, device_num
```

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

omp_target_is_present [25.2.1] [18.8.3]
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, intent(in) :: ptr
integer(c_int), value :: device_num
```

omp_target_is_accessible [25.2.2] [18.8.4]
Tests whether specific *ptr* of memory address and size 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_int, c_ptr, c_size_t
type(c_ptr), value, intent(in) :: ptr
integer(c_size_t), value :: size
integer(c_int), value :: device_num
```

omp_get_mapped_ptr [25.2.3] [18.8.11]
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, intent(in) :: ptr
integer(c_int), value :: device_num
```

omp_target_alloc [25.3] [18.8.1]
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 [25.4] [18.8.2]
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
```

omp_target_associate_ptr [25.5] [18.8.9]
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_int, c_ptr, c_size_t
type(c_ptr), value, intent(in) :: host_ptr, device_ptr
integer(c_size_t), value :: size, device_offset
integer(c_int), value :: device_num
```

omp_target_disassociate_ptr [25.6] [18.8.10]
Removes the associated device data on device *device_num* from the presence table for host printer *ptr*.

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, intent(in) :: ptr
integer(c_int), value :: device_num
```

Continued

Runtime Library Routines (continued)

omp_target_memcpy [25.7.1] [18.8.5]
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_int, c_ptr, c_size_t
type(c_ptr), value :: dst
type(c_ptr), value, intent(in) :: 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 [25.7.2] [18.8.6]
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_int, c_ptr, c_size_t
type(c_ptr), value :: dst
type(c_ptr), value, intent(in) :: 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 [25.7.3] [18.8.7]
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_int, c_ptr, c_size_t
type(c_ptr), value :: dst
type(c_ptr), value, intent(in) :: 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 [25.7.4] [18.8.8]
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_int, c_ptr, c_size_t
type(c_ptr), value :: dst
type(c_ptr), value, intent(in) :: 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_memset [25.8.1]
Fills first *count* bytes indicated by *ptr* with value *val* in the device environment of device associated to *device_num*.

C/C++
```
void *omp_target_memset (void *ptr, int val,
    size_t count, int device_num);
```

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

omp_target_memset_async [25.8.2]
Asynchronously fills the first *count* bytes indicated by *ptr* with the value *val* in the device environment of device associated to *num_device*.

C/C++
```
void *omp_target_memset_async (void *ptr, int val,
    size_t count, int device_num, int depobj_count,
    omp_depobj_t *depobj_list );
```

Fortran
```
type (c_ptr) function omp_target_memset_async  &
    (ptr, val, count, device_num, depobj_count,  &
    depobj_list) bind (c)
use, intrinsic :: iso_c_binding, only :  &
    c_int, c_ptr, c_size_t
type (c_ptr), value :: ptr
integer (c_int), value :: val, device_num, depobj_count
integer (c_size_t), value :: count
integer (omp_depobj_kind), optional :: depobj_list (*)
```

Interoperability routines

omp_get_num_interop_properties [26.1] [18.12.1]
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);
```

Fortran
```
integer function (omp_get_num_interop_properties) &
    (interop)
integer (omp_interop_kind), intent (in) :: interop
```

omp_get_interop_int [26.2] [18.12.2]
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 );
```

Fortran
```
integer (c_intptr_t) function omp_get_interop_int  &
    (interop, property_id, ret_code)
use, intrinsic :: iso_c_binding, only : c_intptr_t
integer (omp_interop_kind), intent(in) :: interop
integer (omp_interop_property_kind) property_id
integer (omp_interop_rc_kind), intent(out), &
    optional :: ret_code
```

omp_get_interop_ptr [26.3] [18.12.3]
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 );
```

Fortran
```
integer (c_ptr) function omp_get_interop_ptr  &
    (interop, property_id, ret_code)
use, intrinsic :: iso_c_binding, only : c_ptr
integer (omp_interop_kind), intent(in) :: interop
integer (omp_interop_property_kind) property_id
integer (omp_interop_rc_kind), intent(out), &
    optional :: ret_code
```

omp_get_interop_str [26.4] [18.12.4]
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 );
```

Fortran
```
character(:) function omp_get_interop_str  &
    (interop, property_id, ret_code)
pointer :: omp_get_interop_str
integer (omp_interop_kind), intent(in) :: interop
integer (omp_interop_property_kind) property_id
integer (omp_interop_rc_kind), intent(out), &
    optional :: ret_code
```

omp_get_interop_name [26.5] [18.12.5]
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);
```

Fortran
```
character(:) function omp_get_interop_name  &
    (interop, property_id)
pointer :: omp_get_interop_name
integer (omp_interop_kind), intent(in) :: interop
integer (omp_interop_property_kind) property_id
```

omp_get_interop_type_desc [26.6] [18.12.6]
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);
```

Fortran
```
character(:) function omp_get_interop_type_desc  &
    (interop, property_id)
pointer :: omp_get_interop_type_desc
integer (omp_interop_kind), intent(in) :: interop
integer (omp_interop_property_kind) property_id
```

Continued

Runtime Library Routines (continued)

omp_get_interop_rc_desc [26.7] [18.12.7]
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);
```
Fortran
```
character(:) function omp_get_interop_rc_desc  &
    (interop, ret_code)
pointer :: omp_get_interop_rc_desc
integer (omp_interop_kind), intent(in) :: interop
integer (omp_interop_rc_kind) ret_code
```

Memory space routines

omp_get_devices_memspace [27.1.1]
This is a memory-space-retrieving routine for the devices specified in the *devs* argument.

C/C++
```
omp _memspace_handle_t
    omp_get_devices_memspace (int ndevs,
    const int *devs,
    omp _memspace_handle_t memspace);
```
Fortran
```
integer (omp_memspace_handle_kind)  &
    function omp_get_devices_memspace   &
    (ndevs, devs, memspace)
integer, intent (in) :: ndevs, devs (*)
integer (omp_memspace_handle_kind),  &
    intent (in) :: memspace
```

omp_get_device_memspace [27.1.2]
The device selected is the device specified in the *dev* argument.

C/C++
```
omp _memspace_handle_t
    omp_get_device_memspace (int dev,
    omp _memspace_handle_t memspace);
```
Fortran
```
integer (omp_memspace_handle_kind)  &
    function omp_get_device_memspace  &
    (dev, memspace)
integer, intent (in) :: dev
integer (omp_memspace_handle_kind),  &
    intent (in) :: memspace
```

omp_get_devices_and_host_memspace [27.1.3]
This is a memory-space-retrieving routine for the host device and devices specified by the *devs* argument.

C/C++
```
omp _memspace_handle_t
    omp_get_devices_and_host_memspace (
    int ndevs, const int *devs,
    omp _memspace_handle_t memspace);
```
Fortran
```
integer (omp_memspace_handle_kind)  &
    function omp_get_devices_and_host_memspace &
    (ndevs, devs, memspace)
integer, intent (in) :: ndevs, devs (*)
integer (omp_memspace_handle_kind),  &
    intent (in) :: memspace
```

omp_get_device_and_host_memspace [27.1.4]
This is a memory-space-retrieving routine for the host device and device specified by the *dev* argument.

C/C++
```
omp _memspace_handle_t
    omp_get_device_and_host_memspace (
    int dev, omp _memspace_handle_t memspace);
```
Fortran
```
integer (omp_memspace_handle_kind)  &
    function omp_get_device_and_host_memspace &
    (dev, memspace)
integer, intent (in) :: dev
integer (omp_memspace_handle_kind),  &
    intent (in) :: memspace
```

omp_get_devices_all_memspace [27.1.5]
This is a memory-space-retrieving routine that selects all available devices.

C/C++
```
omp _memspace_handle_t
    omp_get_devices_all_memspace (
    omp _memspace_handle_t memspace);
```
Fortran
```
integer (omp_memspace_handle_kind)  &
    function omp_get_devices_all_memspace &
    (memspace)
integer (omp_memspace_handle_kind),  &
    intent (in) :: memspace
```

Memory management routines

omp_get_memspace_num_resources [27.2]
Returns the number of distinct storage resources associated with the memory space represented by *memspace*.

C/C++
```
int omp_get_memspace_num_resources (
    omp_memspace_handle_t memspace);
```
Fortran
```
integer function omp_get_memspace_num_resources &
    (memspace)
integer (omp_memspace_handle_kind),  &
    intent (in) :: memspace
```

omp_get_memspace_pagesize [27.3]
Returns the page size that the memory space represented by the *memspace* handle supports.

C/C++
```
omp_intptr_t  omp_get_memspace_pagesize (
    omp_memspace_handle_t memspace);
```
Fortran
```
integer (c_size_t) function  &
    omp_get_memspace_pagesize (memspace) bind(c)
use, intrinsic :: iso_c_binding, only : c_size_t
integer (omp_memspace_handle_kind),  &
    intent (in) :: memspace
```

omp_get_submemspace [27.4]
Returns a new memory space that contains a subset of the resources of the original memory space.

C/C++
```
omp_memspace_handle_t  omp_get_submemspace (
    omp_memspace_handle_t memspace,
    int num_resources, const int *resources);
```
Fortran
```
integer (omp_memspace_handle_kind) function  &
    omp_get_submemspace (memspace, &
    num_resources, resources)
integer (omp_memspace_handle_kind),  &
    intent (in) :: memspace
integer, intent (in) :: num_resources, resources (*)
```

omp_init_allocator [27.6] [18.13.2]
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 (omp_allocator_handle_kind) function &
        omp_init_allocator (memspace, ntraits, traits
integer (omp_memspace_handle_kind),  &
    intent (in) :: memspace
integer, intent (in) :: ntraits
type (omp_alloctrait), intent (in) :: traits (*)
```

omp_destroy_allocator [27.7] [18.13.3]
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 (omp_allocator_handle_kind),  &
    intent (in) :: allocator
```

omp_set_default_allocator [27.9] [18.13.4]
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 (omp_allocator_handle_kind),  &
    intent (in) :: allocator
```

omp_get_default_allocator [27.10] [18.13.5]
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);
```
For
```
integer (omp_allocator_handle_kind) &
    function omp_get_default_allocator ()
```

Memory partitioning routines

omp_init_mempartitioner [27.5.1]
Initializes the memory partitioner that the *partitioner* object represents, with a lifetime (*lifetime*), computation procedure (*compute_proc*), and a release procedure (*release_proc*).

C/C++
```
void omp_init_mempartitioner (
    omp_mempartitioner_t *partitioner,
    omp_mempartitioner_lifetime_t lifetime,
    omp_mempartitioner_compute_proc_t
        compute_proc,
    omp_mempartitioner_release_proc_t release_proc);
```
Fortran
```
subroutine omp_init_mempartitioner ( &
    partitioner, lifetime, compute_proc, release_proc
integer (omp_mempartitioner_kind), &
        intent(out) :: partitioner
integer (omp_mempartitioner_lifetime_kind) &
        intent(in) :: lifetime
procedure (omp_mempartitioner_compute_proc_t) &
        compute_proc
procedure (omp_mempartitioner_release_proc_t) &
        release_proc
```

omp_destroy_mempartitioner [27.5.2]
Uninitializes a memory partitioner, unitializing the state of the memory partitioner and releasing resources.

C/C++
```
void omp_destroy_mempartitioner (
    const omp_mempartitioner_t *partitioner);
```
For
```
subroutine omp_destroy_mempartitioner (partitioner)
integer (omp_mempartitioner_kind), &
        intent(in) :: partitioner
```

omp_init_mempartition [27.5.3]
Initializes a memory partition object *partition* with *nparts* parts and associates the *user_data* argument with it.

C/C++
```
void omp_init_mempartition (
    omp_mempartition_t *partition),
    size_t nparts, const void *user_data);
```
Fortran
```
subroutine omp_init_mempartition (partition, nparts,&
    user_data) bind(c)
use, intrinsic :: iso_c_binding, only : c_ptr, c_size_t
integer (omp_mempartition_kind), &
        intent(out) :: partition
integer (c_size_t),  intent(in) :: nparts
type (c_ptr),  intent(in) :: user_data
```

omp_destroy_mempartition [27.5.4]
Uninitializes a memory partition object *partition*, releasing the memory partition and its resources.

C/C++
```
void omp_destroy_mempartition (
    const omp_mempartition_t *partition);
```
For
```
subroutine omp_destroy_mempartition (partition
integer (omp_mempartition_kind), &
        intent(in) :: partition
```

Continued

Runtime Library Routines (continued)

omp_mempartition_set_part [27.5.5]
Defines the size and resource of a given part of a memory partition.

C/C++
```
int omp_mempartition_set_part (
    omp_mempartition_t *partition,
    size_t *part, int resource, size_t size);
```
Fortran
```
integer function omp_mempartition_set_part &
    (partition, part, resource, size) bind(c)
use, intrinsic :: iso_c_binding, only : c_size_t
integer (omp_mempartition_kind), &
        intent(out) :: partition
integer (c_size_t), intent(in) :: part, size
integer, intent(in) :: resource
```

omp_mempartition_get_user_data [27.5.6]
Retrieves and returns the user data that was associated with the memory partition when it was created.

C/C++
```
void *omp_mempartition_get_user_data (
    const omp_mempartition_t *partition);
```
Fortran
```
type (c_ptr) function  &
    omp_mempartition_get_user_data (partition)  &
    bind(c)
use, intrinsic :: iso_c_binding, only : c_ptr
integer (omp_mempartition_kind), &
        intent(in) ::  partition
```

Memory allocator retrieving routines

omp_get_devices_allocator [27.8.1]
A memory-allocator-retrieving routine for the devices specified in the *devs* argument.

C/C++
```
omp_allocator_handle_t omp_get_devices_allocator
    (int ndevs, const int *devs,
    omp_memspace_handle_t memspace);
```
Fortran
```
integer (omp_allocator_handle_kind) function &
    omp_get_devices_allocator (ndevs, devs, memspace
integer, intent(in) :: ndevs, devs (*)
integer (omp_memspace_handle_kind),  &
    intent(in) :: memspace
```

omp_get_device_allocator [27.8.2]
Get a memory allocator for the device *dev*.

C/C++
```
omp_allocator_handle_t omp_get_device_allocator (
    int dev, omp_memspace_handle_t memspace);
```
Fortran
```
integer (omp_allocator_handle_kind) function &
    omp_get_device_allocator (dev, memspace
integer, intent(in) :: dev
integer (omp_memspace_handle_kind),  &
    intent(in) :: memspace
```

omp_get_devices_and_host_allocator [27.8.3]
Get a memory allocator for the host device and the devices specified in *devs*.

C/C++
```
omp_allocator_handle_t
    omp_get_devices_and_host_allocator (
    int ndevs, const int *devs,
    omp_memspace_handle_t memspace);
```
Fortran
```
integer (omp_allocator_handle_kind) function &
    omp_get_devices_and_host_allocator  &
    (ndevs, devs, memspace)
integer, intent(in) :: ndevs, devs (*)
integer (omp_memspace_handle_kind),  &
    intent(in) :: memspace
```

omp_get_device_and_host_allocator [27.8.4]
Get a memory allocator for the host device and the device *dev*.

C/C++
```
omp_allocator_handle_t
    omp_get_device_and_host_allocator (
    int dev, omp_memspace_handle_t memspace);
```
Fortran
```
integer (omp_allocator_handle_kind) function &
    omp_get_device_and_host_allocator  &
    (dev, memspace)
integer, intent(in) :: dev
integer (omp_memspace_handle_kind),  &
    intent(in) :: memspace
```

omp_get_devices_all_allocator [27.8.5]
Get a memory allocator for all available devices.

C/C++
```
omp_allocator_handle_t
    omp_get_devices_all_allocator (
    omp_memspace_handle_t memspace);
```
Fortran
```
integer (omp_allocator_handle_kind) function &
    omp_get_devices_all_allocator (memspace
integer (omp_memspace_handle_kind),  &
    intent(in) :: memspace
```

Memory allocating routines

omp_alloc and omp_aligned_alloc [27.11.1–2] [18.13.6]
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 omp_null_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_calloc and omp_aligned_calloc [27.11.3–4] [18.13.8]
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
```

omp_realloc [27.11.5] [18.13.9]
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
```

omp_free [27.12] [18.13.7]
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
```

Lock routines
General-purpose lock routines. Two types of locks are supported: simple locks (*svar*) and nestable locks (*nvar*). 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 [28.1.1–2] [18.9.1]

C/C++
```
void omp_init_lock (omp_lock_t *svar);
void omp_init_nest_lock (omp_nest_lock_t *nvar);
```
Fortran
```
subroutine omp_init_lock (svar)
integer (omp_lock_kind) svar
subroutine omp_init_nest_lock (nvar)
integer (omp_nest_lock_kind) nvar
```

Initialize lock with hint [28.1.3–4] [18.9.2]

C/C++
```
void omp_init_lock_with_hint (
    omp_lock_t *svar,
    omp_sync_hint_t hint);
void omp_init_nest_lock_with_hint (
    omp_nest_lock_t *nvar,
    omp_sync_hint_t hint);
```
Fortran
```
subroutine omp_init_lock_with_hint (svar, hint)
integer (omp_lock_kind) svar
integer (omp_sync_hint_kind) hint
subroutine omp_init_nest_lock_with_hint (nvar, hint)
integer (omp_nest_lock_kind) nvar
integer (omp_sync_hint_kind) hint
```
hint: See [20.9.5][15.1] in the specification.

Destroy lock [28.2] [18.9.3]
Ensure that the OpenMP lock is uninitialized.

C/C++
```
void omp_destroy_lock (omp_lock_t *svar);
void omp_destroy_nest_lock (omp_nest_lock_t *nvar);
```
Fortran
```
subroutine omp_destroy_lock (svar)
integer (omp_lock_kind) svar
subroutine omp_destroy_nest_lock (nvar)
integer (omp_nest_lock_kind) nvar
```

Continued

Runtime Library Routines (continued)

Set lock [28.3] [18.9.4]
Sets an OpenMP lock. The calling task region is suspended until the lock is set.

C/C++
```
void omp_set_lock (omp_lock_t *svar);
void omp_set_nest_lock (omp_nest_lock_t *nvar);
```
Fortran
```
subroutine omp_set_lock (svar)
integer (omp_lock_kind) svar
subroutine omp_set_nest_lock (nvar)
integer (omp_nest_lock_kind) nvar
```

Unset lock [28.4] [18.9.5]
Unsets an OpenMP lock or decreases the nesting count.

C/C++
```
void omp_unset_lock (omp_lock_t *svar);
void omp_unset_nest_lock (omp_nest_lock_t *nvar);
```
Fortran
```
subroutine omp_unset_lock (svar)
integer (omp_lock_kind) svar
subroutine omp_unset_nest_lock (nvar)
integer (omp_nest_lock_kind) nvar
```

Test lock [28.5] [18.9.6]
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 *svar);
int omp_test_nest_lock (omp_nest_lock_t *nvar);
```
Fortran
```
logical function omp_test_lock (svar)
integer (omp_lock_kind) svar
integer function omp_test_nest_lock (nvar)
integer (omp_nest_lock_kind) nvar
```

Thread affinity routines

omp_get_proc_bind [29.1] [18.3.1]
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);
```
For
```
integer (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 [29.2] [18.3.2]
Returns the number of places available to the execution environment in the place list.

C/C++
```
int omp_get_num_places (void);
```
For
```
integer function omp_get_num_places ()
```

omp_get_place_num_procs [29.3] [18.3.3]
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 [29.4] [18.3.4]
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 [29.5] [18.3.5]
Returns the place number of the place to which the encountering thread is bound.

C/C++
```
int omp_get_place_num (void);
```
For
```
integer function omp_get_place_num ()
```

omp_get_partition_num_places [29.6] [18.3.6]
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);
```
For
```
integer function omp_get_partition_num_places ()
```

omp_get_partition_place_nums [29.7] [18.3.7]
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 [29.8] [18.3.8]
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);
```
For
```
subroutine omp_set_affinity_format (format)
character(len=*), intent(in) :: format
```

omp_get_affinity_format [29.9] [18.3.9]
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);
```
For
```
integer function omp_get_affinity_format (buffer)
character(len=*), intent(out) :: buffer
```

omp_display_affinity [29.10] [18.3.10]
Prints the OpenMP thread affinity information using the format specification provided.

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

omp_capture_affinity [29.11] [18.3.11]
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
```

Execution control routines

omp_get_cancellation [30.1] [18.2.8]
Returns true if cancellation is enabled; otherwise it returns false. ICV: *cancel-var*

C/C++
```
int omp_get_cancellation (void);
```
For
```
logical function omp_get_cancellation ()
```

omp_pause_resource [30.2.1] [18.6.1]
omp_pause_resource_all [30.2.2] [18.6.2]
Allows the runtime to relinquish resources used by OpenMP on the specified device. Valid *kind* values include, e.g., **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 (omp_pause_resource_kind) kind
integer device_num
integer function omp_pause_resource_all (kind)
integer (omp_pause_resource_kind) kind
```

omp_display_env [30.4] [18.15]
Displays the OpenMP version number and the initial values of ICVs associated with environment variables.

C/C++
```
void omp_display_env (int verbose);
```
For
```
subroutine omp_display_env (verbose)
logical, intent(in) :: verbose
```

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 [30.3.1] [18.10.1]
Returns elapsed wall clock time in seconds.

C/C++
```
double omp_get_wtime (void);
```
For
```
double precision function omp_get_wtime ()
```

omp_get_wtick [30.3.2] [18.10.2]
Returns the precision of the timer (seconds between ticks) used by **omp_get_wtime**.

C/C++
```
double omp_get_wtick (void);
```
For
```
double precision function omp_get_wtick ()
```

Tool support routine

omp_control_tool [31.1] [18.14]
Enables a program to pass commands to an active tool.

C/C++
```
omp_control_tool_result_t omp_control_tool (
    omp_control_tool_t command, int modifier,
    void *arg);
```
Fortran
```
integer (omp_control_tool_result_kind function &
    omp_control_tool (command, modifier)
integer (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 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* [4.3.5] [21.2.5]
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] [21.2].

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 *allocator* [4.4.1] [21.5.1]
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 8.1 6.1
- Allocator traits are listed in Table 8.2 6.2
- Predefined allocators are listed in Table 8.3 6.3

Examples

```
export OMP_ALLOCATOR=omp_high_bw_mem_alloc

export OMP_ALLOCATOR="omp_large_cap_mem_space: \
    alignment=16, pinned=true"

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

Memory space names [Table 8.1 6.1]

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 8.2 6.2]

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 implementation defined)
fallback	**default_mem_fb**, **null_fb**, **abort_fb**, **allocator_fb**
fb_data	An allocator handle (No default)
pinned	**true**, **false** (Default is **false**)
partition	**environment**, **nearest**, **blocked**, **interleaved** (Default is **environment**)
pin_device	Conforming device number (No default)
preferred_device	Conforming device number (No default)
target_access	**single**, **multiple** (Default is **single**)
atomic_scope	**all**, **device** (Default is **device**)
part_size	Positive integer value (Default implementation defined)
partitioner	A memory partitioner handle (No default)
partitioner_arg	An integer value (Default is 0)

Predefined allocators, memory space, and trait values [Table 8.3 6.3]

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_AVAILABLE_DEVICES *list* [4.3.7]
Sets the *available-devices-var* ICV and determines the available non-host devices and their device numbers. The value of *list* must be a comma-separated list of trait specifications or *, where * represents all remaining devices. Each list item expands to a set of matching devices that are supported and accessible and do not match a prior list item. Device numbers are assigned to the resulting devices in order.

OMP_CANCELLATION *cancelstate* [4.3.6] [21.2.6]
Sets the initial value of the *cancel-var* ICV to *cancelstate*, which 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 *debugstate* [4.6.1] [21.4.1]
Sets the *debug-var* ICV to *debugstate*. The value must be **enabled** or **disabled**. If **enabled**, the OpenMP implementation collects 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* [4.3.8] [21.2.7]
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* [4.3.4] [21.2.4]
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* [4.7] [21.7]
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* [4.1.2] [21.1.1]
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* [4.1.5] [21.1.4]
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* [4.3.11] [21.2.9]
Sets the initial value of the *max-task-priority-var* ICV that controls the use of task priorities.

OMP_NUM_TEAMS *number* [4.2.1] [21.6.1]
Sets the maximum number of teams created by a **teams** construct by setting the *nteams-var* ICV.

OMP_NUM_THREADS *list* [4.1.3] [21.1.2]
Sets the initial value of the *nthreads-var* ICV for the number of threads to use for **parallel** regions.

OMP_PLACES *places* [4.1.6] [21.1.6]
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.

OMP_PROC_BIND *policy* [4.1.7] [21.1.7]
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]* [4.3.1] [21.2.1]
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[unit]* [4.3.2] [21.2.2]
Sets the *stacksize-var* ICV that specifies the size of the stack for threads created by the OpenMP implementation. The value of *size* is a positive integer that specifies stack size. The value of *unit* indicates the unit of measurement, where **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 *state* [4.3.9] [21.2.8]
Sets the initial value of the *target-offload-var* ICV. The value of *state* must be one of **mandatory**, **disabled**, or **default**.

OMP_TEAMS_THREAD_LIMIT *number* [4.2.2] [21.6.2]
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* [4.1.4] [21.1.3]
Sets the maximum number of OpenMP threads to use in a contention group by setting the *thread-limit-var* ICV.

OMP_THREADS_RESERVE *list* [4.3.10]
Controls the number of structured threads and free-agent threads to reserve in a contention group by accordingly setting the *structured-thread-limit-var* and *free-agent-thread-limit-var* ICVs. The value of *list* is a comma-separated list of reservations, where a reservation is **n**, **structured**[(*n*)]), or **free_agent**[(*n*)] and where *n* is a non-negative integer.

OMP_TOOL *toolstate* [4.5.1] [21.3.1]
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. The value of *toolstate* may be **enabled** or **disabled**.

OMP_TOOL_LIBRARIES *library-list* [4.5.2] [21.3.2]
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 *value* [4.5.3] [21.3.3]
Sets the *tool-verbose-init-var* ICV, which controls whether an OpenMP implementation will verbosely log the registration of a tool. The value of *value* must be a filename or one of **disabled**, **stdout**, or **stderr**.

OMP_WAIT_POLICY *policy* [4.3.3] [21.2.3]
Sets the *wait-policy-var* ICV that provides a hint to an OpenMP implementation about the desired behavior of waiting threads. *policy* may be **active** (waiting threads consume processor cycles while waiting) or **passive**. The 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 3.1-3] [Tables 2.1–3]

ICV	Environment variable	Initial value	Ways to modify value	Ways to retrieve value	Scope
active-levels-var	(none)	*zero*	(none)	**omp_get_active_level()**	Data env.
affinity-format-var	**OMP_AFFINITY_FORMAT** [4.3.5] [21.2.5]	Implementation defined.	**omp_set_affinity_format()**	**omp_get_affinity_format()**	Device
available-devices-var	**OMP_AVAILABLE_DEVICES** [4.3.7]	The set of all supported and accessible devices.	(none)	(none)	Global
bind-var	**OMP_PROC_BIND** [4.1.7] [21.1.7]	Implementation defined.	(none)	**omp_get_proc_bind()**	Data env.
cancel-var	**OMP_CANCELLATION** [4.3.6] [21.2.6]	False	(none)	**omp_get_cancellation()**	Global
debug-var	**OMP_DEBUG** [4.6.1] [21.4.1]	disabled	(none)	(none)	Global
def-allocator-var	**OMP_ALLOCATOR** [4.4.1] [21.5.1]	Implementation defined.	**omp_set_default_allocator()**	**omp_get_default_allocator()**	Impl. Task
default-device-var	**OMP_DEFAULT_DEVICE** [4.3.8] [21.2.7]	Implementation defined.	**omp_set_default_device()**	**omp_get_default_device()**	Data env.
device-num-var	(none)	Zero	(none)	**omp_get_device_num()**	Device
display-affinity-var	**OMP_DISPLAY_AFFINITY** [4.3.4] [21.2.4]	False	(none)	(none)	Global
dyn-var	**OMP_DYNAMIC** [4.1.2] [21.1.1]	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.
explicit-task-var	(none)	False	(none)	**omp_in_explicit_task()**	Data env.
final-task-var	(none)	False	(none)	**omp_in_final()**	Data env.
free-agent-thread-limit-var	**OMP_THREAD_LIMIT** [4.1.4] [21.1.3] **OMP_THREADS_RESERVE** [4.3.10]	One less than the initial *value of thread-limit-var*.	(none)	(none)	Data env.
free-agent-var	(none)	*False*	(none)	**omp_is_free_agent()**	Data env.
league-size-var	(none)	*One*	(none)	**omp_get_num_teams()**	Data env.
levels-var	(none)	*Zero*	(none)	**omp_get_level()**	Data env.
max-active-levels-var	**OMP_MAX_ACTIVE_LEVELS** [4.1.5] [21.1.4] **OMP_NUM_THREADS** [4.1.3] [21,1,2] **OMP_PROC_BIND** [4.1.7] [21.1.7]	Implementation defined.	**omp_set_max_active_levels()**	**omp_get_max_active_levels()**	Data env.
max-task-priority-var	**OMP_MAX_TASK_PRIORITY** [4.3.11] [21.2.9]	*Zero*	(none)	**omp_get_max_task_priority()**	Global
nteams-var	**OMP_NUM_TEAMS** [4.2.1] [21.6.1]	*Zero*	**omp_set_num_teams()**	**omp_get_max_teams()**	Device
nthreads-var	**OMP_NUM_THREADS** [4.1.3] [21.1.2]	Implementation defined.	**omp_set_device_num_teams()** **omp_set_num_threads()**	**omp_get_device_num_teams()** **omp_get_max_threads()**	Data env.
num-devices-var	(none)	Implementation defined.	(none)	**omp_get_num_devices()**	Global
num-procs-var	(none)	Implementation defined.	(none)	**omp_get_num_procs()**	Device
place-assignment-var	(none)	Implementation defined.	(none)	(none)	Impl. Task
place-partition-var	**OMP_PLACES** [4.1.6] [21.1.6]	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
run-sched-var	**OMP_SCHEDULE** [4.3.1] [21.2.1]	Implementation defined.	**omp_set_schedule()**	**omp_get_schedule()**	Data env.
stacksize-var	**OMP_STACKSIZE** [4.3.2] [21.2.2]	Implementation defined.	(none)	(none)	Device
structured-thread-limit-var	**OMP_THREAD_LIMIT** [4.1.4] [21.1.3] **OMP_THREADS_RESERVE** [4.3.10]	The initial value of thread-limit-var.	(none)	(none)	Data env.
target-offload-var	**OMP_TARGET_OFFLOAD** [4.3.9] [21.2.8]	default	(none)	(none)	Global
team-generator-var	(none)	*Zero*	(none)	(none)	Data env.
team-num-var	(none)	*Zero*	(none)	**omp_get_team_num()**	Data env.
team-size-var	(none)	*One*	(none)	**omp_get_num_threads()**	Data env.
teams-thread-limit-var	**OMP_TEAMS_THREAD_LIMIT** [4.2.2] [21.6.2]	*Zero*	**omp_set_device_teams_thread_limit()** **omp_set_teams_thread_limit()**	**omp_get_device_teams_thread_limit()** **omp_get_teams_thread_limit()**	Device
thread-limit-var	**OMP_THREAD_LIMIT** [4.1.4] [21.1.3]	Implementation defined.	**thread_limit** clause at **target** and **teams** constructs	**omp_get_thread_limit()**	Data env.
thread-num-var	(none)	*Zero*	(none)	**omp_get_thread_num()**	Data env.
tool-libraries-var	**OMP_TOOL_LIBRARIES** [4.5.2] [21.3.2]	Empty string	(none)	(none)	Global
tool-var	**OMP_TOOL** [4.5.1] [21.3.1]	Enabled	(none)	(none)	Global
tool-verbose-init-var	**OMP_TOOL_VERBOSE_INIT** [4.5.3] [21.3.3]	Disabled	(none)	(none)	Global
wait-policy-var	**OMP_WAIT_POLICY** [4.3.3] [21.2.3]	Implementation defined.	(none)	(none)	Device

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.

Become an OpenMP ARB Member

The strength of the OpenMP Architecture Review Board (ARB) comes from its diverse membership.

Companies both large and small with products that depend on the OpenMP API are invited to join the OpenMP ARB, as are other organizations with interest in the evolution of the OpenMP API.

OpenMP ARB provides members with Developer Resources, API Influence, Access to the API Roadmap, Co-marketing Opportunities, OpenMP Community Participation , and the right to use OpenMP trademarks and logos to promote your OpenMP API-related products, services, and research.

Learn more at openmp.org/join

OpenMP API 6.0 Reference Guide Index