OpenMP 5.0 API Syntax Reference Guide

openmp.org

The OpenMP® API is a portable, scalable model that gives parallel programmers a simple and flexible interface for developing portable parallel applications in C/C++ and Fortran. OpenMP is suitable for a wide range of algorithms running on multicore nodes and chips, NUMA systems, GPUs, and other such devices attached to a CPU.

C/C++ | C/C++ content
For | Fortran content

Functionality new/changed in OpenMP 5.0 is in this color, and in OpenMP 4.5 is in this color.
[n.n.n] Sections in the 5.0 spec. [n.n.n] Sections in the 4.5 spec. ● Deprecated in the 5.0 spec.

Directives and Constructs

An OpenMP executable directive applies to the succeeding structured block. A *structured-block* is an OpenMP construct or a block of executable statements with a single entry at the top and a single exit at the bottom. OpenMP directives except SIMD and **declare target** directives may not appear in **PURE** or **ELEMENTAL** procedures.

variant directives

Metadirectives [2.3.4]

A directive that can specify multiple directive variants of which one 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
```

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

clause:
- **when** (*context-selector-specification*: [*directive-variant*])
- **default** (*directive-variant*)

declare variant [2.3.5]

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

C/C++
```
#pragma omp declare variant(variant-func-id) clause
[#pragma omp declare variant(variant-func-id) clause]
[ ... ]
    function definition or declaration
```

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

clause:
- **match** (*context-selector-specification*)

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

variant-proc-name: For
The name of a function variant that is a base language identifier.

requires directive

requires [2.4]

Specifies the features 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:
- **reverse_offload**
- **unified_address**
- **unified_shared_memory**
- **atomic_default_mem_order(seq_cst | acq_rel | relaxed)**
- **dynamic_allocators**

parallel construct

parallel [2.6] [2.5]

Creates a team of OpenMP threads that execute the region.

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

For
```
!$omp parallel [clause[ [, ]clause] ...]
    structured-block
!$omp end parallel
```

clause:
- **private** (*list*), **firstprivate** (*list*), **shared** (*list*)
- **copyin** (*list*)
- **reduction** ([*reduction-modifier*,] *reduction-identifier*: *list*)
- **proc_bind** (**master** | **close** | **spread**)
- **allocate** ([*allocator* :]*list*)
- C/C++ **if** ([**parallel** :] *scalar-expression*)
- C/C++ **num_threads** (*integer-expression*)
- C/C++ **default** (**shared** | **none**)
- For **if** ([**parallel** :] *scalar-logical-expression*)
- For **num_teams** (*scalar-integer-expression*)
- For **default** (**shared** | **firstprivate** | **private** | **none**)

clause:
- **private** (*list*), **firstprivate** (*list*)
- **lastprivate** ([*lastprivate-modifier:*] *list*)
- **reduction** ([*reduction-modifier,*] *reduction-identifier*: *list*)
- **allocate** ([*allocator* :]*list*)
- C/C++ **nowait**

teams construct

teams [2.7] [2.10.7]

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

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

For
```
!$omp teams [clause[ [, ]clause] ...]
    structured-block
!$omp end teams
```

clause:
- **private** (*list*), **firstprivate** (*list*), **shared** (*list*)
- **reduction** ([**default** ,] *reduction-identifier* : *list*)
- **allocate** ([*allocator* :]*list*)
- C/C++ **num_teams** (*integer-expression*)
- C/C++ **thread_limit** (*integer-expression*)
- C/C++ **default** (**shared** | **none**)
- For **num_teams** (*scalar-integer-expression*)
- For **thread_limit** (*scalar-integer-expression*)
- For **default** (**shared** | **firstprivate** | **private** | **none**)

Worksharing constructs

sections [2.8.1] [2.7.2]

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

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

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

single [2.8.2] [2.7.3]

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
```

For
```
!$omp single [clause[ [, ]clause] ...]
    structured-block
!$omp end single [end_clause[ [, ]end_clause] ...]
```

clause:
- **private** (*list*), **firstprivate** (*list*)
- **allocate** ([*allocator* :]*list*)
- C/C++ **copyprivate** (*list*)
- C/C++ **nowait**

end_clause: For
- **copyprivate** (*list*), **nowait**

workshare [2.8.3] [2.7.4]

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

For
```
!$omp workshare
    structured-block
!$omp end workshare [nowait]
```

Worksharing-loop construct

for / do [2.9.2] [2.7.1]

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

C/C++
```
#pragma omp for [clause[ [, ]clause] ...]
    for-loops
```

For
```
!$omp do [clause[ [, ]clause] ...]
    do-loops
[!$omp end do [nowait] ]
```

clause:
- **private** (*list*), **firstprivate** (*list*)
- **lastprivate** ([*lastprivate-modifier:*] *list*)
- **linear** (*list*[: *linear-step*])
- **schedule** ([*modifier* [, *modifier*] :] *kind*[, *chunk_size*])
- **collapse** (*n*), **ordered** [(*n*)]
- **allocate** ([*allocator* :]*list*)
- **order (concurrent)**
- C/C++ **reduction** ([*reduction-modifier,*] *reduction-identifier*: *list*)
- C/C++ **nowait**

kind:
- **static:** Iterations are divided into chunks of size *chunk_size* and assigned to threads in the team 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:** Each thread executes a chunk of iterations then requests another chunk until no chunks remain to be assigned. Chunk size is different for each chunk, with each successive chunk smaller than the last.

Continued

© 2018 OpenMP ARB

Directives and Constructs (continued)

- **auto:** The decision regarding scheduling is delegated to the compiler and/or runtime system.
- **runtime:** The schedule and chunk size are taken from the *run-sched-var* ICV.

modifier:

- **monotonic:** Each thread executes the chunks that it is assigned in increasing logical iteration order. Default for **static** schedule.
- **nonmonotonic:** Chunks are assigned to threads in any order and the behavior of an application that depends on execution order of the chunks is unspecified. Default for all schedule kinds except **static**.
- **simd:** Ignored when the loop is not associated with a SIMD construct, otherwise the *new_chunk_size* for all except the first and last chunks is ⌈*chunk_size*/*simd_width*⌉ * *simd_width* where *simd_width* is an implementation-defined value.

SIMD directives

simd [2.9.3.1] [2.8.1]

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

C/C++:
```
#pragma omp simd [clause[ [, ]clause] ...]
    for-loops
```
For:
```
!$omp simd [clause[ [, ]clause] ...]
    do-loops
[!$omp end simd]
```

clause:

safelen (*length***)**, **simdlen (***length***)**
linear (*list*[: *linear-step*]**)**
aligned (*list*[: *alignment*]**)**
nontemporal (*list***)**
private (*list***)**
lastprivate ([*lastprivate-modifier* :] *list***)**
reduction ([*reduction-modifier*,] *reduction-identifier* **:** *list***)**
collapse (*n***)**
order (concurrent)
C/C++ **if (**[*simd* :] *scalar-expression***)**
For **if (**[*simd* :] *scalar-logical-expression***)**

Worksharing-Loop SIMD [2.9.3.2] [2.8.3]

Applied to a loop to indicate that the loop can be transformed into a SIMD loop that will be executed in parallel by threads in the team.

C/C++:
```
#pragma omp for simd [clause[ [, ]clause] ...]
    for-loops
```
For:
```
!$omp do simd [clause[ [, ]clause] ...]
    do-loops
[!$omp end do simd[nowait] ]
```

clause: Any of the clauses accepted by the **simd** or **do** directives with identical meanings and restrictions.

declare simd [2.9.3.3] [2.8.2]

Applied to a function or a subroutine to enable the creation of one or more versions that can process multiple arguments using SIMD instructions from a single invocation from 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]
    ...]
```

clause:

simdlen (*length***)**
linear (*linear-list*[: *linear-step*]**)**
aligned (*argument-list*[**:** *alignment*]**)**
uniform (*argument-list***)**
inbranch
notinbranch

distribute loop constructs

distribute [2.9.4.1] [2.10.8]

Specifies loops which are executed by the thread teams.

C/C++:
```
#pragma omp distribute [clause[ [, ]clause] ...]
    for-loops
```
For:
```
!$omp distribute [clause[ [, ]clause] ...]
    do-loops
[!$omp end distribute]
```

clause:

private (*list***)**
firstprivate (*list***)**
lastprivate (*list***)**
collapse (*n***)**
dist_schedule (*kind*[, *chunk_size*]**)**
allocate ([*allocator* :]*list***)**

distribute simd [2.9.4.2] [2.10.9]

Specifies loops which are executed concurrently using SIMD instructions and the thread teams.

C/C++:
```
#pragma omp distribute simd [clause[ [, ]clause] ...]
    for-loops
```
For:
```
!$omp distribute simd [clause[ [, ]clause] ...]
    do-loops
[!$omp end distribute simd]
```

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

Distribute Parallel Worksharing-Loop [2.9.4.3] [2.10.10]

These constructs 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] ...]
    for-loops
```
For:
```
!$omp distribute parallel do [clause[ [, ]clause] ...]
    do-loops
[!$omp end distribute parallel do]
```

clause: Any accepted by the **distribute** or parallel worksharing-loop directives with identical meanings and restrictions.

Distribute Parallel Worksharing-Loop SIMD [2.9.4.4] [2.10.11]

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] ...]
    for-loops
```
For:
```
!$omp distribute parallel do simd [clause[ [, ]clause]
    ...]
    do-loops
[!$omp end distribute parallel do simd]
```

clause: Any accepted by the **distribute** or parallel worksharing-loop SIMD directives with identical meanings and restrictions.

loop construct

loop [2.9.5]

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] ...]
    for-loops
```
For:
```
!$omp loop [clause[ [, ]clause] ...]
    do-loops
[!$omp end loop ]
```

clause:

bind (*binding***)**
collapse (*n***)**
order (concurrent)
private (*list***)**
lastprivate (*list***)**
reduction ([**default** ,]*reduction-identifier* **:** *list***)**

binding:

teams, parallel, thread

scan directive

scan [2.9.6]

Specifies that scan computations update the list items on each iteration.

C/C++:
```
loop-associated-directive
for-loop-headers
{
    structured-block
    #pragma omp scan clause
    structured-block
}
```
For:
```
loop-associated-directive
do-loop-headers
    structured-block
    !$omp scan clause
    structured-block
do-termination-stmts(s)
[end-loop-associated-directive ]
```

clause:

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

loop-directive directives: C/C++

for, for simd, simd directive

[end-]loop-associated-directive directives: For

do (end do)
do simd (end do simd)
simd (end simd)

Tasking constructs

task [2.10.1] [2.9.1]

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

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

clause:

untied, mergeable
private (*list***), firstprivate (***list***), shared (***list***)**
in_reduction (*reduction-identifier: list***)**
depend ([*depend-modifier*,] *dependence-type* **:** *locator-list***)**
priority(*priority-value***)**
allocate([*allocator:*]*list***)**
affinity ([*aff-modifier:*] *locator-list***)**
  - where *aff-modifier* is **iterator(***iterators-definition***)**
detach (*event-handle***)**
  - where *event-handle* is of type **omp_event_handle**
C/C++ **default (shared | none)**
C/C++ **if (**[**task :**] *scalar-expression***)**
C/C++ **final (***scalar-expression***)**
For **default (private | firstprivate | shared | none)**
For **if (**[**task :**] *scalar-logical-expression***)**
For **final (***scalar-logical-expression***)**

taskloop [2.10.2] [2.9.2]

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

C/C++:
```
#pragma omp taskloop [clause[ [, ]clause] ...]
    for-loops
```
For:
```
!$omp taskloop [clause[ [, ]clause] ...]
    do-loops
[!$omp end taskloop ]
```

clause:

shared (*list***), private (***list***)**
firstprivate (*list***), lastprivate (***list***)**
reduction ([**default** ,] *reduction-identifier* **:** *list***)**
in_reduction (*reduction-identifier* **:** *list***)**
grainsize (*grain-size***), num_tasks (***num-tasks***)**
collapse (*n***), priority (***priority-value***)**
untied, mergeable, nogroup
allocate ([*allocator:*]*list***)**
C/C++ **if (**[**taskloop :**] *scalar-expression***)**
C/C++ **default (shared | none)**
C/C++ **final (***scalar-expr***)**
For **if (**[**taskloop :**] *scalar-logical-expression***)**
For **default (private | firstprivate | shared | none)**
For **final (***scalar-logical-expr***)**

Continued

Directives and Constructs (continued)

taskloop simd [2.10.3] [2.9.3]

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] ...]* *for-loops*
For	**!$omp taskloop simd** *[clause[[,]clause] ...]* *do-loops* *[***!$omp end taskloop simd***]*

clause: Any accepted by the **simd** or **taskloop** directives with identical meanings and restrictions.

taskyield [2.10.4] [2.11.2]

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

C/C++	**#pragma omp taskyield**
For	**!$omp taskyield**

Memory management directive

Memory spaces [2.11.1]

Predefined memory spaces [Table 2.7, below] 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 optimized for variables with constant values.
omp_high_bw_mem_space	Storage with high bandwidth.
omp_low_lat_mem_space	Storage with low latency.

allocate [2.11.3]

Specifies how a set of variables is allocated.

C/C++	**#pragma omp allocate** (*list*) *[clause]*
For	**!$omp allocate** (*list*) *[clause]* *or* **!$omp allocate** *[* (*list*) *] clause* *[* **!$omp allocate** (*list*) *clause* *[...]]* *allocate statement*

clause:

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

Device directives and construct

target data [2.12.2] [2.10.1]

Creates a device data environment for extent of the region.

C/C++	**#pragma omp target data** *clause[[[,]clause] ...]* *structured-block*
For	**!$omp target data** *clause[[[,]clause] ...]* *structured-block* **!$omp end target data**

clause:

map (*[[map-type-modifier[,] [map-type-modifier[,] ...]*
map-type: *] locator-list*)
use_device_ptr (*list*), **use_device_addr** (*list*)
C/C++ **if** (*[***target data** : *] scalar-expression*)
C/C++ **device** (*scalar-expression*)
For **if** (*[***target data** : *] scalar-logical-expression*)
For **device** (*scalar-integer-expression*)

target enter data [2.12.3] [2.10.2]

Maps variables to a device data environment.

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

clause:

map (*[map-type-modifier[,] [map-type-modifier[,] ...]*
map-type: *locator-list*)
depend (*[depend-modifier,] dependence-type* : *locator-list*)
nowait
C/C++ **if** (*[* **target enter data** : *] scalar-expression*) C/C++
C/C++ **device** (*integer-expression*) C/C++
For **if** (*[* **target enter data** : *] scalar-logical-expression*)
For **device** (*scalar-integer-expression*)

target exit data [2.12.4] [2.10.3]

Unmaps variables from a device data environment.

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

clause:

map (*[map-type-modifier[,] [map-type-modifier[,] ...]*
map-type: *locator-list*)
depend (*[depend-modifier,] dependence-type* :
locator-list)**nowait**
if (*[* **target exit data** : *] scalar-expression*) C/C++
device (*integer-expression*) C/C++
if (*[* **target exit data** : *] scalar-logical-expression*) For
device (*scalar-integer-expression*) For

target [2.12.5] [2.10.4]

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

C/C++	**#pragma omp target** *[clause[[,]clause] ...]* *structured-block*
For	**!$omp target** *[clause[[,]clause] ...]* *structured-block* **!$omp end target**

clause:

if (*[* **target** : *] scalar-expression*)
private (*list*), **firstprivate** (*list*)
in_reduction (*reduction-identifier* : *list*)
map (*[[map-type-modifier[,] [map-type-modifier[,] ...]*
map-type: *] locator-list*)
is_device_ptr (*list*)
defaultmap (*implicit-behavior[:variable-category]*)
nowait
depend(*[depend-modifier,] dependence-type* : *locator-list*)
allocate (*[allocator* : *] list*)
uses_allocators (*allocator[* (*allocator-traits-array*)*]*
[,allocator[(*allocator-traits-array*) *] ...]*)
C/C++ **if** (*[* **target** : *] scalar-expression*)
C/C++ **device**(*[device-modifier:] integer-expression*)
For **if** (*[* **target** : *] scalar-logical-expression*)
For **device** (*[device-modifier:] scalar-integer-expression*)

device-modifier: **ancestor, device_num**

allocator: C/C++
Identifier of type **omp_allocator_handle_t**

allocator: For
Integer expression of *kind* **omp_allocator_handle_kind**

allocator-traits-array: C/C++
Identifier of **const omp_alloctrait_t** * type.

allocator-traits-array: For
Array of **type(omp_alloctrait)** type

target update [2.12.6] [2.10.5]

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: motion-clause or one of:

nowait
depend (*[depend-modifier,] dependence-type* : *locator-list*)
C/C++ **if** (*[* **target update** : *] scalar-expression*)
C/C++ **device** (*integer-expression*)
For **if** (*[* **target update** : *] scalar-logical-expression*)
For **device** (*scalar-integer-expression*)

motion-clause:

to (*[***mapper**(*mapper-identifier*) : *] locator-list*)
from (*[***mapper**(*mapper-identifier*) : *] locator-list*)

declare target [2.12.7] [2.10.6]

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

C/C++	**#pragma omp declare target** *declarations-definition-seq* **#pragma omp end declare target** *- or -* **#pragma omp declare target** (*extended-list*) *- or -* **#pragma omp declare target** *clause[[,]clause ...]*
For	**!$omp declare target** (*extended-list*) *- or -* **!$omp declare target** *[clause[[,]clause] ...]*

clause:

to (*extended-list*), **link** (*list*)
device_type (host | nohost | any)

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

Combined constructs

Parallel Worksharing Loop [2.13.1] [2.11.1]

Specifies a **parallel** construct containing one worksharing-loop construct with one or more associated loops.

C/C++	**#pragma omp parallel for** *[clause[[,]clause] ...]* *for-loop*
For	**!$omp parallel do** *[clause[[,]clause] ...]* *do-loops* *[***!$omp end parallel do***]*

clause: Any accepted by the **parallel** or **for** directives, except the **nowait** clause, with identical meanings and restrictions.

parallel loop [2.13.2]

Shortcut for specifying a **parallel** construct containing **loop** construct with one or more associated loops and no other statements.

C/C++	**#pragma omp parallel loop** *[clause[[,]clause] ...]* *for-loops*
For	**!$omp parallel loop** *[clause[[,]clause] ...]* *do-loops* *[***!$omp end parallel loop***]*

clause: Any accepted by the **parallel** or **loop** directives, with identical meanings and restrictions.

parallel sections [2.13.3] [2.11.2]

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

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

For	**!$omp parallel sections** *[clause[[,]clause] ...]* *[***!$omp section***]* *structured-block* *[***!$omp section** *structured-block]* ... **!$omp end parallel sections**

clause: Any clauses accepted by the **parallel** or **sections** directives, with identical meanings and restrictions.

parallel workshare [2.13.4] [2.11.3]

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

For	**!$omp parallel workshare** *[clause[[,]clause] ...]* *structured-block* **!$omp end parallel workshare**

clause: Any of the clauses accepted by the **parallel** directive, with identical meanings and restrictions.

Continued

Directives and Constructs (continued)

Parallel Worksharing-Loop SIMD [2.13.5] [2.11.4]

Shortcut for specifying a **parallel** construct containing one **do simd** construct and no other statements.

C/C++
```
#pragma omp parallel for simd [clause[ [, ]clause] ...]
    for-loops
```
For
```
!$omp parallel do simd [clause[ [, ]clause] ...]
    do-loops
[!$omp end parallel do simd]
```
clause: Any accepted by the **parallel** or **for/do simd** directives with identical meanings and restrictions.

parallel master [2.13.6]

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

C/C++
```
#pragma omp parallel master [clause[ [, ]clause] ...]
    structured-block
```
For
```
!$omp parallel master [clause[ [, ]clause] ...]
    structured-block
$omp end parallel master
```
clause: Any clause used for **parallel** or **master** directives with identical meanings and restrictions.

master taskloop [2.13.7]

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

C/C++
```
#pragma omp master taskloop [clause[ [, ]clause] ...]
    for-loops
```
For
```
!$omp master taskloop [clause[ [, ]clause] ...]
    do-loops
[$omp end master taskloop]
```
clause: Any clause used for **master** or **taskloop** directives with identical meanings and restrictions.

master taskloop simd [2.13.8]

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

C/C++
```
#pragma omp master taskloop simd \
        [clause[ [, ]clause] ...]
    for-loops
```
For
```
!$omp master taskloop simd [clause[ [, ]clause] ...]
    do-loops
[$omp end master taskloop simd]
```
clause: Any clause used for **master** or **taskloop simd** directives with identical meanings and restrictions.

parallel master taskloop [2.13.9]

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

C/C++
```
#pragma omp parallel master taskloop \
        [clause[ [, ]clause] ...]
    for-loops
```
For
```
!$omp parallel master taskloop [clause[ [, ]clause] ...]
    do-loops
[$omp end parallel master taskloop]
```
clause: Any clause used for **parallel** or **master taskloop** directives, except the **in_reduction** clause, with identical meanings and restrictions.

parallel master taskloop simd [2.13.10]

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

C/C++
```
#pragma omp parallel master taskloop simd \
        [clause[ [, ]clause] ...]
    for-loops
```
For
```
!$omp parallel master taskloop simd [clause[ [, ]clause] ...]
    do-loops
[$omp end parallel master taskloop simd]
```
clause: Any clause used for **parallel** or **master taskloop simd** directives, except the **in_reduction** clause, with identical meanings and restrictions.

teams distribute [2.13.11] [2.11.10]

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

C/C++
```
#pragma omp teams distribute [clause[ [, ]clause] ...]
    for-loops
```
For
```
!$omp teams distribute [clause[ [, ]clause] ...]
    do-loops
[ !$omp end teams distribute ]
```
clause: Any accepted by the **teams** or **distribute** directives with identical meanings and restrictions.

teams distribute simd [2.13.12] [2.11.11]

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

C/C++
```
#pragma omp teams distribute simd [clause[ [, ]
        clause] ...]
    for-loops
```
For
```
!$omp teams distribute simd [clause[ [, ]clause] ...]
    do-loops
[!$omp end teams distribute simd]
```
clause: Any accepted by the **teams** or **distribute simd** directives with identical meanings and restrictions.

Teams Distribute Parallel Worksharing-Loop [2.13.13] [2.11.14]

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] ...]
    for-loops
```
For
```
!$omp teams distribute parallel do [clause[ [, clause] ...]
    do-loops
[ !$omp end teams distribute parallel do ]
```
clause: Any clause used for **teams** or **distribute parallel do/for** with identical meanings and restrictions.

Teams Distribute Parallel Worksharing-Loop SIMD [2.13.14] [2.11.7]

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

C/C++
```
#pragma omp teams distribute parallel for simd \
        [clause[ [, ]clause] ...]
    for-loops
```
For
```
!$omp teams distribute parallel do simd
        [clause[ [, ]clause] ...]
    do-loops
[!$omp end teams distribute parallel do simd]
```
clause: Any accepted by **teams** or **distribute parallel for/do simd**, with identical meanings and restrictions.

teams loop [2.13.15]

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

C/C++
```
#pragma omp teams loop [clause[ [, ]clause] ...]
    for -loops
```
For
```
!$omp teams loop [clause[ [, ]clause] ...]
    do-loops
[!$omp end teams loop]
```
clause: Any accepted by the **teams** or **loop** directives with identical meanings and restrictions.

target parallel [2.13.16] [2.11.5]

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

C/C++
```
#pragma omp target parallel [clause[ [, ]clause] ...]
    structured-block
```
For
```
!$omp target parallel [clause[ [, ]clause] ...]
    structured-block
[!$omp end target parallel]
```
clause: Any accepted by the **target** or **parallel** directives, except for **copyin**, with identical meanings and restrictions.

Target Parallel Worksharing-Loop [2.13.17] [2.11.8]

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

C/C++
```
#pragma omp target parallel for [clause[ [, ]clause] ...]
    for -loops
```
For
```
!$omp target parallel do [clause[ [, ]clause] ...]
    do-loops
[!$omp end target parallel do]
```
clause: Any accepted by the **target** or **parallel for/do** directives, except for **copyin**, with identical meanings and restrictions.

Target Parallel Worksharing-Loop SIMD [2.13.18] [2.11.9]

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

C/C++
```
#pragma omp target parallel for simd \
        [clause[ [, ]clause] ...]
    for-loops
```
For
```
!$omp target parallel for simd [clause[ [, ]clause] ...]
    do-loops
!$omp end target parallel for simd
```
clause: Any accepted by the **target** or **parallel for/do simd** directives with identical meanings and restrictions.

target parallel loop [2.13.19] [2.11.10]

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

C/C++
```
#pragma omp target parallel loop [clause[ [, ]clause] ...]
    for-loops
```
For
```
!$omp target parallel loop [clause[ [, ]clause] ...]
    do-loops
[ !$omp end target parallel loop ]
```
clause: Any accepted by the **teams** or **parallel loop** directives with identical meanings and restrictions.

target simd [2.13.20] [2.11.8]

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

C/C++
```
#pragma omp target simd [clause[ [, ]clause] ...]
    for-loops
```
For
```
!$omp target simd [clause[ [, ]clause] ...]
    do-loops
[!$omp end target simd]
```
clause: Any accepted by the **target** or **simd** directives with identical meanings and restrictions.

target teams [2.13.21] [2.11.9]

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

C/C++
```
#pragma omp target teams [clause[ [, ]clause] ...]
    structured-block
```
For
```
!$omp target teams [clause[ [, ]clause] ...]
    structured-block
[!$omp end target teams]
```
clause: Any accepted by the **target** or **teams** directives with identical meanings and restrictions.

target teams distribute [2.13.22] [2.11.12]

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

C/C++
```
#pragma omp target teams distribute [clause[ [, ]
        clause] ...]
    for-loops
```
For
```
!$omp target teams distribute [clause[ [, ]clause] ...]
    do-loops
[!$omp end target teams distribute]
```
clause: Any accepted by the **target** or **teams distribute** directives with identical meanings and restrictions.

Continued

© 2018 OpenMP ARB

Directives and Constructs (continued)

target teams distribute simd [2.13.23] [2.11.13]

Shortcuts 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] ...]
  for-loops
```

For
```
!$omp target teams distribute simd [clause[ [, ]clause] ...]
  do-loops
[ !$omp end target teams distribute simd ]
```

clause: Any accepted by the **target** or **teams distribute simd** directives with identical meanings and restrictions.

Target Teams Loop [2.13.24]

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

C/C++
```
#pragma omp target teams loop [clause[ [, ]clause] ...]
  for-loops
```

For
```
!$omp target teams loop [clause[ [, ]clause] ...]
  do-loops
[ !$omp end target teams loop ]
```

clause: Any clause used for **target** or **teams loop** directives with identical meanings and restrictions.

Target Teams Distribute Parallel Worksharing-Loop [2.13.25] [2.11.15]

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

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

For
```
!$omp target teams distribute parallel do &
    [clause[ [, ]clause] ...]
  do-loops
[$omp end target teams distribute parallel do]
```

clause:
Any clause used for **teams distribute parallel do** or **teams distribute parallel for** or **target** directives with identical meanings and restrictions.

Target Teams Distribute Parallel Worksharing-Loop SIMD [2.13.26] [2.11.17]

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] ...]
  for-loops
```

For
```
!$omp target teams distribute parallel do simd &
    [clause[ [, ]clause] ...]
  do-loops
[!$omp end target teams distribute parallel do simd]
```

clause: Any clause used for **teams distribute parallel do simd**, **teams distribute parallel for simd**, or **target** directives with identical meanings and restrictions.

master construct

master [2.16] [2.13.1]

Specifies a structured block that is executed by the master thread of the team.

C/C++
```
#pragma omp master
  structured-block
```

For
```
!$omp master
  structured-block
!$omp end master
```

Synchronization constructs

critical [2.17.1] [2.13.2]

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
```

For
```
!$omp critical [(name) [[,] hint (hint-expression)] ]
  structured-block
!$omp end critical [(name)]
```

hint-expression: C/C++
An integer constant expression that evaluates to a valid synchronization hint

hint-expression: For
A constant expression that evaluates to a scalar value with kind **omp_sync_hint_kind** and a value that is a valid synchronization hint

barrier [2.17.2] [2.13.3]

Placed only at a point where a base language statement is allowed, this directive specifies an explicit barrier at the point at which the construct appears.

C/C++
```
#pragma omp barrier
```

For
```
!$omp barrier
```

taskwait [2.17.5] [2.13.4]

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***)**

taskgroup [2.17.6] [2.13.5]

Specifies a wait on the completion of child tasks of the current task, and waits for descendant tasks.

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

For
```
!$omp taskgroup [clause[ [, ]clause] ...]
  structured-block
!$omp end taskgroup
```

clause:
task_reduction (*reduction-identifier* : *list***)**
allocate ([*allocator*:]*list***)**

atomic [2.17.7] [2.13.6]

Ensures a specific storage location is accessed atomically. May take one of the following seven forms:

C/C++
```
#pragma omp atomic [clause[[[,] clause] ... ] [,]]
    atomic-clause
  [[,] clause [[[,] clause] ...]]
  expression-stmt

#pragma omp atomic [clause[[,] clause] ...]
  expression-stmt

#pragma omp atomic [clause[[[,] clause] ... ] [,] capture
  [[,] clause [[[,] clause] ... ]]
  structured-block
```

For
```
!$omp atomic [clause[[[,] clause] ... ] [,]] read &
  [[,] clause [[[,] clause] ...]]
  capture-statement
[!$omp end atomic]

!$omp atomic [clause[[[,] clause] ... ] [,]] write &
  [[,] clause [[[,] clause] ...]]
  write-statement
[!$omp end atomic]

!$omp atomic [clause[[[,] clause] ... ] [,]] update &
  [[,] clause [[[,] clause] ...]]
  update-statement
[!$omp end atomic]

!$omp atomic [clause[[,] clause] ...]
  update-statement
[!$omp end atomic]
```

Continued in next column

For
```
!$omp atomic [clause[[[,] clause] ... ] [,]] capture &
  [[,] clause [[[,] clause] ...]]
  update-statement
  capture-statement
!$omp end atomic

!$omp atomic [clause[[[,] clause] ... ] [,]] capture &
  [[,] clause [[[,] clause] ...]]
  capture-statement
  update-statement
!$omp end atomic

!$omp atomic [clause[[[,] clause] ... ] [,]] capture &
  [[,] clause [[[,] clause] ...]]
  capture-statement
  write-statement
!$omp end atomic
```

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

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

clause: memory-order-clause or **hint (***hint-expression***)**

expression-stmt: C/C++

if *atomic clause* is...	*expression-stmt*:
read	*v = x;*
write	*x = expr;*
update or is not present	*x++;* *x--;* *++x;* *--x;* *x binop= expr;* *x = x binop expr;* *x = expr binop x;*
capture	*v=x++;* *v=x--;* *v=++x;* *v= --x;* *v=x binop= expr;* *v=x = x binop expr;* *v=x = expr binop x;*

structured-block may be one of the following forms: C/C++

{v = x; x binop= expr;} {x binop= expr; v = x;}
{v = x; x = x binop expr;} {v = x; x = expr binop x;}
{x = x binop expr; v = x;} {x = expr binop x; v = x;}
{v = x; x = expr;} {v = x; x++;} {v = x; ++x;}
{++x; v = x;} {x++; v = x;} {v = x; x--;}
{v = x; --x;} {--x; v = x;} {x--; v = x;}

capture-, *write-*, or *update-statement:* For

capture-statement	*v = x*
write-statement	*x = expr*
update-statement	*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 is one of **MAX**, **MIN**, **IAND**, **IOR**, **IEOR**
operator is one of **+**, *****, **-**, **/**, **.AND.**, **.OR.**, **.EQV.**, **.NEQV.**

flush [2.17.8] [2.13.7]

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: **acq_rel**, **release**, or **acquire**

ordered [2.17.9] [2.13.8]

Specifies a structured block in a worksharing-loop, **simd**, or worksharing-loop SIMD region, 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]...]
```

For
```
!$omp ordered [clause[[, ] clause]]
  structured-block
!$omp end ordered
- or -
!$omp ordered clause[[[, ]clause] ...]
```

clause (for the first form): **threads** or **simd**

clause (for the second form):
depend (source) or **depend (sink** : *vec***)**

Continued

Directives and Constructs (continued)

depobj [2.17.10.1]

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

C/C++	**#pragma omp depobj (***depobj***)** *clause*
For	**!$omp depobj (***depobj***)** *clause*

clause:
 depend (*dependence-type* **:** *locator***)**
 destroy
 update (*dependence-type***)**

Cancellation constructs

cancel [2.18.1] [2.14.1]

Requests 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]*

construct-type-clause:
 parallel, sections, taskgroup
 C/C++ **for**
 For **do**

if-clause:
 C/C++ **if (***[* **cancel** *:] scalar-expression***)**
 For **if (***[* **cancel** *:] scalar-logical-expression***)**

cancellation point [2.18.2] [2.14.2]

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**

Data environment directive

threadprivate [2.19.2] [2.15.2]

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

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

declare reduction [2.19.5.7] [2.16]

Declares a *reduction-identifier* that can be used in a **reduction** clause.

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

type-list or *typename-list:* A list of type specifiers

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

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

combiner: C/C++ An expression

reduction-identifier: 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.

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

declare mapper [2.19.7.3]

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] ...]*

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

type: A valid type in scope

var: A valid base-language identifier

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

map-type:
 alloc, to, from, tofrom

map-type-modifier:
 always, close

Runtime Library Routines

Execution environment routines

omp_set_num_threads [3.2.1] [3.2.1]

Affects the number of threads used for subsequent parallel regions 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 [3.2.2] [3.2.2]

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

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

omp_get_max_threads [3.2.3] [3.2.3]

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

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

omp_get_thread_num [3.2.4] [3.2.4]

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

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

omp_get_num_procs [3.2.5] [3.2.5]

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_in_parallel [3.2.6] [3.2.6]

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

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

omp_set_dynamic [3.2.7] [3.2.7]

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 [3.2.8] [3.2.8]

This routine returns the value of the *dyn-var* ICV, which is *true* if dynamic adjustment of the number of threads is enabled for the current task.

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

omp_get_cancellation [3.2.9] [3.2.9]

Returns the value of the *cancel-var* ICV, which is *true* if cancellation is activated; otherwise it returns *false*.

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

● omp_set_nested [3.2.10] [3.2.10]

Enables or disables nested parallelism, by setting the *max-active-levels-var* ICV.

C/C++	**void omp_set_nested (int** *nested***);**
For	**subroutine omp_set_nested (***nested***)** **logical** *nested*

● omp_get_nested [3.2.11] [3.2.11]

Returns whether nested parallelism is enabled or disabled, according to the value of the *max-active-levels-var* ICV.

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

omp_set_schedule [3.2.12] [3.2.12]

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

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

See **omp_get_schedule** for *kind*.

Continued

© 2018 OpenMP ARB

Runtime Library Routines (continued)

omp_get_schedule [3.2.13] [3.2.13]

Returns the value of *run-sched-var* ICV, which is the schedule applied when **runtime** schedule is used.

C/C++:
```
void omp_get_schedule (
    omp_sched_t *kind, int *chunk_size);
```

For:
```
subroutine omp_get_schedule (kind, chunk_size)
integer (kind=omp_sched_kind) kind
integer chunk_size
```

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

- omp_sched_static
- omp_sched_dynamic
- omp_sched_guided
- omp_sched_auto

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

omp_get_thread_limit [3.2.14] [3.2.14]

Returns the value of the *thread-limit-var* ICV, which is the maximum number of OpenMP threads available.

C/C++:
```
int omp_get_thread_limit (void);
```

For:
```
integer function omp_get_thread_limit ()
```

omp_get_supported_active_levels [3.2.15]

Returns the number of active levels of parallelism supported.

C/C++:
```
void omp_get_supported_active_levels( void);
```

For:
```
integer function omp_get_supported_active_levels ()
```

omp_set_max_active_levels [3.2.16] [3.2.15]

Limits the number of nested active parallel regions, 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 [3.2.17] [3.2.16]

Returns the value of *max-active-levels-var* ICV, which determines the maximum number of nested active parallel regions.

C/C++:
```
int omp_get_max_active_levels (void);
```

For:
```
integer function omp_get_max_active_levels ()
```

omp_get_level [3.2.18] [3.2.17]

For the enclosing device region, returns the *levels-vars* ICV, which is the number of nested **parallel** regions that enclose the task containing the call.

C/C++:
```
int omp_get_level (void);
```

For:
```
integer function omp_get_level ()
```

omp_get_ancestor_thread_num [3.2.19] [3.2.18]

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

C/C++:
```
int omp_get_ancestor_thread_num (int level);
```

For:
```
integer function omp_get_ancestor_thread_num (
    level)
integer level
```

omp_get_team_size [3.2.20] [3.2.19]

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

C/C++:
```
int omp_get_team_size (int level);
```

For:
```
integer function omp_get_team_size (level)
integer level
```

omp_get_active_level [3.2.21] [3.2.20]

Returns the value of the *active-level-vars* ICV, which determines the number of active, nested **parallel** regions enclosing the task that contains the call.

C/C++:
```
int omp_get_active_level (void);
```

For:
```
integer function omp_get_active_level ()
```

omp_in_final [3.2.22] [3.2.21]

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_get_proc_bind [3.2.23] [3.2.22]

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 (kind=omp_proc_bind_kind)
    function omp_get_proc_bind ()
```

Returns one of:

- omp_proc_bind_false
- omp_proc_bind_true
- omp_proc_bind_master
- omp_proc_bind_close
- omp_proc_bind_spread

omp_get_num_places [3.2.24] [3.2.23]

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 [3.2.25] [3.2.24]

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);
```

For:
```
integer function omp_get_place_num_procs (
    place_num)
integer place_num
```

omp_get_place_proc_ids [3.2.26] [3.2.25]

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);
```

For:
```
subroutine omp_get_place_proc_ids(place_num, ids)
integer place_num
integer ids (*)
```

omp_get_place_num [3.2.27] [3.2.26]

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 [3.2.28] [3.2.27]

Returns the number of places in the place partition 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 [3.2.29] [3.2.28]

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);
```

For:
```
subroutine omp_get_partition_place_nums(
    place_nums)
integer place_nums (*)
```

omp_set_affinity_format [3.2.30]

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 [3.2.31]

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 [3.2.32]

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 [3.2.33]

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)
```

For:
```
integer function omp_capture_affinity (buffer, format)
character(len=*), intent(out) :: buffer
character(len=*), intent(in) :: format
```

omp_set_default_device [3.2.34] [3.2.29]

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 [3.2.35] [3.2.30]

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

C/C++:
```
int omp_get_default_device (void);
```

For:
```
integer function omp_get_default_device ()
```

omp_get_num_devices [3.2.36] [3.2.31]

Returns the number of target devices.

C/C++:
```
int omp_get_num_devices (void);
```

For:
```
integer function omp_get_num_devices ()
```

omp_get_device_num [3.2.37]

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 ()
```

Continued

Runtime Library Routines (continued)

omp_get_num_teams [3.2.38] [3.2.32]

Returns the number of teams in the current **teams** region, or 1 if called from outside of a **teams** region.

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

omp_get_team_num [3.2.39] [3.2.33]

Returns the team number of the calling thread. The team number is an integer between 0 and one less than the value returned by **omp_get_num_teams**, inclusive.

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

omp_is_initial_device [3.2.40] [3.2.34]

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	integer function **omp_is_initial_device** ()

omp_get_initial_device [3.2.41] [3.2.35]

Returns a device number representing the host device.

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

omp_get_max_task_priority [3.2.42] [3.2.36]

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_pause_resource [3.2.43]

Allows the runtime to relinquish resources used by OpenMP on the specified device.

C/C++	int **omp_pause_resource** (**omp_pause_resource_t** *kind*, int *device_num*);
For	integer function **omp_pause_resource** (**kind, device_num**) integer (**kind=omp_pause_resource_kind**) *kind* integer **device_num**

omp_pause_resource_all [3.2.44]

Allows the runtime to relinquish resources used by OpenMP on the specified device.

C/C++	int **omp_pause_resource_all** (**omp_pause_resource_t** *kind*);
For	integer function **omp_pause_resource_all** (*kind*) integer (**kind=omp_pause_resource_kind**) *kind*

Lock routines

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

Initialize lock [3.3.1] [3.3.1]

Initialize an OpenMP lock.

C/C++	void **omp_init_lock** (omp_lock_t **lock*); void **omp_init_nest_lock** (omp_nest_lock_t **lock*);
For	subroutine **omp_init_lock** (*svar*) integer (**kind=omp_lock_kind**) *hint* subroutine **omp_init_nest_lock** (*nvar*) integer (**kind=omp_nest_lock_kind**) *nvar*

Initialize lock with hint [3.3.2] [3.3.2]

Initialize an OpenMP lock with a hint.

C/C++	void **omp_init_lock_with_hint** (omp_lock_t **lock*, omp_sync_hint_t *hint*); void **omp_init_nest_lock_with_hint** (omp_nest_lock_t **lock*, omp_sync_hint_t *hint*);
For	subroutine **omp_init_lock_with_hint** (*svar, hint*) integer (**kind=omp_lock_kind**) *svar* integer (**kind=omp_sync_hint_kind**) *hint* subroutine **omp_init_nest_lock_with_hint** (*nvar, hint*) integer (**kind=omp_nest_lock_kind**) *nvar* integer (**kind=omp_sync_hint_kind**) *hint*

hint: see [2.17.12]

Destroy lock [3.3.3] [3.3.3]

Ensure that the OpenMP lock is uninitialized.

C/C++	void **omp_destroy_lock** (omp_lock_t **lock*); void **omp_destroy_nest_lock** (omp_nest_lock_t **lock*);
For	subroutine **omp_destroy_lock** (*svar*) integer (**kind=omp_lock_kind**) *svar* subroutine **omp_destroy_nest_lock** (*nvar*) integer (**kind=omp_nest_lock_kind**) *nvar*

Set lock [3.3.4] [3.3.4]

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

C/C++	void **omp_set_lock** (omp_lock_t **lock*); void **omp_set_nest_lock** (omp_nest_lock_t **lock*);
For	subroutine **omp_set_lock** (*svar*) integer (**kind=omp_lock_kind**) *svar* subroutine **omp_set_nest_lock** (*nvar*) integer (**kind=omp_nest_lock_kind**) *nvar*

Unset lock [3.3.5] [3.3.5]

Unsets an OpenMP lock.

C/C++	void **omp_unset_lock** (omp_lock_t **lock*); void **omp_unset_nest_lock** (omp_nest_lock_t **lock*);
For	subroutine **omp_unset_lock** (*svar*) integer (**kind=omp_lock_kind**) *svar* subroutine **omp_unset_nest_lock** (*nvar*) integer (**kind=omp_nest_lock_kind**) *nvar*

Test lock [3.3.6] [3.3.6]

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

C/C++	int **omp_test_lock** (omp_lock_t **lock*); int **omp_test_nest_lock** (omp_nest_lock_t **lock*);
For	logical function **omp_test_lock** (*svar*) integer (**kind=omp_lock_kind**) *svar* integer function **omp_test_nest_lock** (*nvar*) integer (**kind=omp_nest_lock_kind**) *nvar*

Timing routines

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

omp_get_wtime [3.4.1] [3.4.1]

Returns elapsed wall clock time in seconds.

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

omp_get_wtick [3.4.2] [3.4.2]

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** ()

Event routine

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

omp_fulfill_event [3.5.1]

Fulfills and destroys an OpenMP event.

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

Device memory routines

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

omp_target_alloc [3.6.1] [3.5.1]

Allocates memory in a device data environment.

C/C++	void ***omp_target_alloc** (size_t *size*, int *device_num*);

omp_target_free [3.6.2] [3.5.2]

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

C/C++	void **omp_target_free** (void **device_ptr*, int *device_num*);

omp_target_is_present [3.6.3] [3.5.3]

Validates whether a host pointer has an associated device buffer on a given device.

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

omp_target_memcpy [3.6.4] [3.5.4]

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*);

omp_target_memcpy_rect [3.6.5] [3.5.5]

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*);

omp_target_associate_ptr [3.6.6] [3.5.6]

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*);

omp_target_disassociate_ptr [3.6.7] [3.5.7]

Removes the associated pointer for a given device from a host pointer.

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

Continued

© 2018 OpenMP ARB

Runtime Library Routines (continued)

Memory management routines

Memory Management Types [3.7.2]

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

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

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

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

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

omp_init_allocator [3.7.2]

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*[]);
For	**integer (kind=omp_allocator_handle_kind) &** **function omp_init_allocator (&** *memspace*, *ntraits*, *traits*) **integer (kind=omp_memspace_handle_kind), &** **intent (in)** :: *memspace* **integer, intent (in)** :: *ntraits* **&** **type (omp_alloctrait), intent (in)** :: *traits* **(*)**

omp_destroy_allocator [3.7.3]

Releases all resources used by the allocator handle.

C/C++	void **omp_destroy_allocator (** omp_allocator_handle_t *allocator*);
For	**subroutine omp_destroy_allocator** (*allocator*) **integer (kind=omp_allocator_handle_kind), &** **intent (in)** :: *allocator*

omp_set_default_allocator [3.7.4]

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*);
For	**subroutine omp_set_default_allocator** (*allocator*) **integer (kind=omp_allocator_handle_kind), &** **intent (in)** :: *allocator*

omp_get_default_allocator [3.7.5]

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 (kind=omp_allocator_handle_kind)** **function omp_get_default_allocator ()**

omp_alloc [3.7.6]

Requests a memory allocation from a memory allocator.

C	void ***omp_alloc** (size_t *size*, omp_allocator_handle_t *allocator*);
C++	void ***omp_alloc** (size_t *size*, const omp_allocator_t *allocator*=omp_null_allocator);

omp_free [3.7.7]

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);

Tool control routine

omp_control_tool [3.8]

Enables a program to pass commands to an active tool.

C/C++	int **omp_control_tool** (int *command*, int *modifier*, void **arg*);
For	**integer function omp_control_tool (&** *command*, *modifier*) **integer (kind=omp_control_tool_kind)** *command* **integer** *modifier*

command:

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

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

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

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

Clauses

All list items appearing in a clause must be visible according to the scoping rules of the base language. Not all of the clauses listed in this section are valid on all directives.

Allocate Clause [2.11.4]

allocate ([*allocator:*] *list***)**

Specifies the memory allocator to be used to obtain storage for private variables of a directive.

allocator:
C/C++ Expression of type **omp_allocator_handle_t**
For Integer expression of *kind* **omp_allocator_handle_kind**

Data Copying Clauses [2.19.6] [2.15.4]

copyin (*list***)**
Copies the value of the master thread's threadprivate variable to the threadprivate variable of each other member of the team executing the **parallel** region.

copyprivate (*list***)**
Broadcasts a value from the data environment of one implicit task to the data environments of the other implicit tasks belonging to the **parallel** region.

Data Sharing Attribute Clauses [2.19.4] [2.15.3]

Applies only to variables whose names are visible in the construct on which the clause appears.

default (shared | none) *C/C++*

default (private | firstprivate | shared | none) *For*

Explicitly determines default data-sharing attributes of variables referenced in a **parallel**, **teams**, or task generating construct, causing all variables referenced in the construct that have implicitly determined data-sharing attributes to be as specified.

shared (*list***)**
Declares list items to be shared by tasks generated by **parallel**, **teams**, or task-generating construct. Storage shared by explicit **task** region must not reach the end of its lifetime before the explicit task region completes execution.

private (*list***)**
Declares list items to be private to a task or a SIMD lane. Each task or SIMD lane that references a list item in the construct receives only one new list item, unless the construct has one or more associated loops and the **order(concurrent)** clause is also present.

firstprivate (*list***)**
Declares list items to be private to a task, and initializes each of them with the value that the corresponding original item has when the construct is encountered.

lastprivate ([*lastprivate-modifier:*] *list***)**
Declares one or more list items to be private to an implicit task or SIMD lane, and causes the corresponding original list item to be updated after the end of the region.

lastprivate-modifier: **conditional**

linear (*linear-list*[*: linear-step*]**)**
Declares one or more list items to be private and to have a linear relationship with respect to the iteration space of a loop associated with the construct on which the clause appears.

linear-list: list or *modifier*(*list*)

modifier: **ref**, **val**, or **uval** (*C: modifier* may only be **val**)

Defaultmap Clause [2.19.7.2] [2.15.5.2]

defaultmap (*implicit-behavior*[*: variable-category*]**)**
Explicitly determines the data-mapping attributes referenced in a **target** construct and would otherwise be implicitly determined.

map-type: **alloc**, **to**, **from**, **tofrom**, **firstprivate**, **none**, **default**

variable-category: *C/C++*
scalar, **aggregate**, **pointer**

variable-category:
scalar, **aggregate**, **pointer, allocatable** *For*

Depend Clause [2.17.11] [2.13.9]

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

depend (*dependence-type***)**
dependence-type may be source.

depend (*dependence-type : vec***)**
dependence-type: sink and is the iteration vector with the form: x1 [± d1], x2 [± d2], . . . , xn [± dn]

depend ([*depend-modifier,*]*dependence-type* **:** *locator-list***)**
dependence-modifier: **iterator** (*iterators-definition*)
dependence-type: **in**, **out**, **inout**, **mutexinoutset**, **depobj**

- **in**: The generated task will be dependent of all previously generated sibling tasks that reference at least one of the list items in an **out** or **inout** *dependence-type* list.

Continued

© 2018 OpenMP ARB

Clauses (continued)

- **out** and **inout**: The generated task will be dependent of all previously generated sibling tasks that reference at least one of the list items in an **in**, **out**, **mutexinoutset**, or **inout** *dependence-type* list.
- **mutexinoutset**: If the storage location of at least one of the list items is the same as that of a list item appearing in a **depend** clause with an **in**, **out**, or **inout** *dependence-type* on a construct from which a sibling task was previously generated, then the generated task will be a dependent task of that sibling task. If the storage location of at least one of the list items is the same as that of a list item appearing in a **depend** clause with a **mutexinoutset** *dependence-type* on a construct from which a sibling task was previously generated, then the sibling tasks will be mutually exclusive tasks.

Depend (continued)

- **depobj**: The task dependences are derived from the **depend** clause specified in the **depobj** constructs that initialized dependences represented by the depend objects specified on in the **depend** clause as if the **depend** clauses of the **depobj** constructs were specified in the current construct.

If Clause [2.15] [2.12]

The effect of the **if** clause depends on the construct to which it is applied. For combined or composite constructs, it only applies to the semantics of the construct named in the *directive-name-modifier* if one is specified. If none is specified for a combined or composite construct then the **if** clause applies to all constructs to which an **if** clause can apply.

if ([*directive-name-modifier* :] *scalar-expression***)** C/C++

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

Map Clause [2.19.7.1] [2.15.5.1]

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

Map an original list item from the current task's data environment to a corresponding list item in the device data environment of the device identified by the construct.

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

map-type-modifier: **always**, **close**, **mapper** (*mapper-identifier*)

Reduction Clauses [2.19.5]

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

Specifies a *reduction-identifier* and one or more list items.

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

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

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

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

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

Specifies a reduction among tasks.

reduction-identifier: Same as for **reduction**

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

Specifies that a task participates in a reduction.

reduction-identifier: Same as for **reduction**

SIMD Clauses [2.9.3] [2.8]

safelen (*length***)**

If used then no two iterations executed concurrently with SIMD instructions can have a greater distance in the logical iteration space than its value.

collapse (*n***)**

A constant positive integer expression that specifies how many loops are associated with the construct.

simdlen (*length***)**

A constant positive integer expression that specifies the number of concurrent arguments of the function.

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

Declares one or more list items to be aligned to the specified number of bytes. *alignment*, if present, must be a constant positive integer expression.

uniform (*argument-list***)**

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

inbranch

Specifies that the function will always be called from inside a conditional statement of a SIMD loop.

notinbranch

Specifies that the function will never be called from inside a conditional statement of a SIMD loop.

Tasking Clauses [2.10] [2.9]

affinity ([*aff-modifier*:] *locator-list***)**

A hint to execute closely to the location of the list items. *aff-modifier* is **iterator (***iterators-definition***)**.

allocate ([*allocator*:]*list***)**

See Allocate Clause, page 9 of this guide.

collapse (*n***)**

See SIMD Clauses on this page.

default (shared | **none)** C/C++

default (private | **firstprivate** | **shared** | **none)** For

See Data Sharing Attribute Clauses, page 9 of this guide.

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

See Depend Clause, page 9 of this guide.

final (*scalar-expression***)** C/C++

final (*scalar-logical-expression***)** For

The generated task will be a final task if the final expression evaluates to true.

firstprivate (*list***)**

See Data Sharing Attribute Clauses, page 9 of this guide.

grainsize (*grain-size***)**

Causes the number of logical loop iterations assigned to each created task to be greater than or equal to the minimum of the value of the *grain-size* expression and the number of logical loop iterations, but less than two times the value of the *grain-size* expression.

if ([**task :**] *scalar-expression***)** C/C++

if ([**task :**] *scalar-logical-expression***)** For

Also see If Clause on this page.

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

See Reduction Clause on this page.

lastprivate (*list***)**

See Data Sharing Attribute Clauses, page 9 of this guide.

mergeable

Specifies that the generated task is a mergeable task.

nogroup

Prevents an implicit **taskgroup** region to be created.

num_tasks (*num-tasks***)**

Create as many tasks as the minimum of the *num-tasks* expression and the number of logical loop iterations.

priority (*priority-value***)**

A non-negative numerical scalar expression that specifies a hint for the priority of the generated task.

private (*list***)**

See Data Sharing Attribute Clauses, page 9 of this guide.

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

See Reduction Clauses on this page.

shared (*list***)**

See Data Sharing Attribute Clauses, page 9 of this guide.

untied

If present, any thread in the team can resume the task region after a suspension.

Iterators

iterators [2.1.6]

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

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

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

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

identifier: A base language identifier.

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

step: An integral expression.

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

iterator-type: A type specifier. For

Notes

© 2018 OpenMP ARB

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 the host device are modified accordingly. The method for initializing a target device's ICVs is implementation defined.

Table of ICV Initial Values (Table 2.1) and Ways to Modify and to Retrieve ICV Values (Table 2.2) [2.5.2-3] [2.3.2-3]

ICV	Environment variable	Initial value	Ways to modify value	Ways to retrieve value	Env. Var. Ref.
dyn-var	**OMP_DYNAMIC**	Implementation-defined if the implementation supports dynamic adjustment of the number of threads; otherwise, the initial value is *false*.	**omp_set_dynamic()**	**omp_get_dynamic()**	[6.3] [4.3]
● *nest-var*	● **OMP_NESTED**	Implementation defined.	● **omp_set_nested()**	● **omp_get_nested()**	[6.9] [4.6]
nthreads-var	**OMP_NUM_THREADS**	Implementation defined list.	**omp_set_num_threads()**	**omp_get_max_threads()**	[6.2] [4.2]
run-sched-var	**OMP_SCHEDULE**	Implementation defined.	**omp_set_schedule()**	**omp_get_schedule()**	[6.1] [4.1]
def-sched-var	(none)	Implementation defined.	(none)	(none)	---
bind-var	**OMP_PROC_BIND**	Implementation defined list.	(none)	**omp_get_proc_bind()**	[6.4] [4.4]
stacksize-var	**OMP_STACKSIZE**	Implementation defined.	(none)	(none)	[6.6] [4.7]
wait-policy-var	**OMP_WAIT_POLICY**	Implementation defined.	(none)	(none)	[6.7] [4.8]
thread-limit-var	**OMP_THREAD_LIMIT**	Implementation defined.	**thread_limit** *clause*	**omp_get_thread_limit()**	[6.10] [4.10]
max-active-levels-var	**OMP_MAX_ACTIVE_LEVELS**, ● **OMP_NESTED**	The number of levels of parallelism that the implementation supports.	**omp_set_max_active_levels()**, ● **omp_set_nested()**	**omp_get_max_active_levels()**	[6.8] [4.9
active-levels-var	(none)	*zero*	(none)	**omp_get_active_level()**	---
levels-var	(none)	*zero*	(none)	**omp_get_level()**	---
place-partition-var	**OMP_PLACES**	Implementation defined.	(none)	**omp_get_partition_num_places()** **omp_get_partition_place_nums()** **omp_get_place_num_procs()** **omp_get_place_proc_ids()**	[6.5] [4.5]
cancel-var	**OMP_CANCELLATION**	*false*	(none)	**omp_get_cancellation()**	[6.11] [4.11]
display-affinity-var	**OMP_DISPLAY_AFFINITY**	*false*	(none)	(none)	[6.13]
affinity-format-var	**OMP_AFFINITY_FORMAT**	Implementation defined.	**omp_set_affinity_format()**	**omp_get_affinity_format()**	[6.14]
default-device-var	**OMP_DEFAULT_DEVICE**	Implementation defined.	**omp_set_default_device()**	**omp_get_default_device()**	[6.15] [4.13]
target-offload-var	**OMP_TARGET_OFFLOAD**	**DEFAULT**	(none)	(none)	[6.17]
max-task-priority-var	**OMP_MAX_TASK_PRIORITY**	*zero*	(none)	**omp_get_max_task_priority()**	[6.16] [4.14]
tool-var	**OMP_TOOL**	*enabled*	(none)	(none)	[6.18]
tool-libraries-var	**OMP_TOOL_LIBRARIES**	*empty string*	(none)	(none)	[6.19]
debug-var	**OMP_DEBUG**	*disabled*	(none)	(none)	[6.20]
def-allocator-var	**OMP_ALLOCATOR**	Implementation defined.	**omp_set_default_allocator()**	**omp_get_default_allocator()**	[6.21]

Environment Variables

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

OMP_ALLOCATOR *arg* [6.21]
Sets the *def-allocator-var* ICV that specifies the default allocator for allocation calls, directives and clauses that do not specify an allocator. The *arg* is a case-insensitive, predefined allocator below (for details, see table 2.9):

omp_default_mem_alloc	**omp_low_lat_mem_alloc**
omp_large_cap_mem_alloc	**omp_cgroup_mem_alloc**
omp_const_mem_alloc	**omp_pteam_mem_alloc**
omp_high_bw_mem_alloc	**omp_thread_mem_alloc**

OpenMP memory allocators can be used to make allocation requests. The behavior of the allocation process can be affected by the allocator traits specified. [Table 2.9] below shows allowed allocator traits and their possible values, with the default value shown in blue.

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

OMP_AFFINITY_FORMAT *format* [6.14]
Sets the initial value of the *affinity-format-var* ICV defining the format when displaying OpenMP thread affinity information. The argument is a character string that may contain as substrings one or more field specifiers, in addition to other characters. 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 5.2].

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_CANCELLATION *var* [6.11] [4.11]
Sets the *cancel-var* ICV. *var* may be **true** or **false**. If **true**, the effects of the cancel construct and of cancellation points are enabled and cancellation is activated.

OMP_DEBUG *var* [6.20]
Sets the *debug-var* ICV. var may be **enabled** or **disabled**. If **enabled**, the OpenMP implementation will collect additional runtime information to be provided to a third-party tool. If **disabled**, only reduced functionality might be available in the debugger.

OMP_DEFAULT_DEVICE *device* [6.15] [4.13]
Sets the *default-device-var* ICV that controls the default device number to use in device constructs.

OMP_DISPLAY_AFFINITY *var* [6.13]
Instructs the runtime to display formatted affinity information for all OpenMP threads in the parallel region. The information is displayed upon entering the first parallel region and when there is any change in the information accessible by the format specifiers listed in the table for **OMP_AFFINITY_FORMAT**. If there is a change of affinity of any thread in a parallel region, thread affinity information for all threads in that region will be displayed.

OMP_DISPLAY_ENV *var* [6.12] [4.12]
If *var* is **TRUE**, instructs the runtime to display the OpenMP version number and the value of the ICVs associated with the environment variables as *name=value* pairs. If *var* is **VERBOSE**, the runtime may also display vendor-specific variables. If *var* is **FALSE**, no information is displayed.

OMP_DYNAMIC *var* [6.3] [4.3]
Sets the *dyn-var* ICV. If **TRUE**, the implementation may dynamically adjust the number of threads to use for executing **parallel** regions.

Continued

© 2018 OpenMP ARB

Environment Variables (continued)

OMP_MAX_ACTIVE_LEVELS *levels* [6.8] [4.9]
Sets the *max-active-levels-var* ICV that controls the maximum number of nested active **parallel** regions.

OMP_MAX_TASK_PRIORITY *level* [6.16] [4.14]
Sets the *max-task-priority-var* ICV that controls the use of task priorities.

● **OMP_NESTED** *nested* [6.9] [4.6]
Controls nested parallelism with *max-active-levels-var* ICV.

OMP_NUM_THREADS *list* [6.2] [4.2]
Sets the *nthreads-var* ICV for the number of threads to use for **parallel** regions.

OMP_PLACES *places* [6.5] [4.5]
Sets the *place-partition-var* ICV that defines the OpenMP places available to the execution environment. *places* is an abstract name (**threads**, **cores**, **sockets**, or **implementation-defined**) or a list of non-negative numbers.

OMP_PROC_BIND *policy* [6.4] [4.4]
Sets the value of the global *bind-var* ICV, setting the thread affinity policy to use for parallel regions at the corresponding nested level. *policy* can be the values **true**, **false**, or a comma-separated list of **master**, **close**, or **spread** in quotes.

OMP_SCHEDULE *[modifier:]kind[, chunk]* [6.1] [4.1]
Sets the *run-sched-var* ICV for the runtime schedule kind and chunk size. *modifier* is one of **monotonic** or **nonmonotonic**; *kind* is one of **static**, **dynamic**, **guided**, or **auto**.

OMP_STACKSIZE *size*[**B** | **K** | **M** | **G**] [6.6] [4.7]
Sets the *stacksize-var* ICV that specifies the size of the stack for threads created by the OpenMP implementation. *size* is a positive integer that specifies stack size. If unit is not specified, *size* is measured in kilobytes (K).

OMP_TARGET_OFFLOAD *arg* [6.17]
Sets the initial value of the *target-offload-var* ICV. The argument must be one of **MANDATORY**, **DISABLED**, or **DEFAULT**.

OMP_THREAD_LIMIT *limit* [6.10] [4.10]
Sets the *thread-limit-var* ICV that controls the number of threads participating in the OpenMP program.

OMP_TOOL (**enabled**|**disabled**) [6.18]
Sets the *tool-var* ICV. If disabled, no first-party tool will be loaded nor initialized. If enabled the OpenMP implementation will try to find and activate a first-party tool.

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

OMP_WAIT_POLICY *policy* [6.7] [4.8]
Sets the *wait-policy-var* ICV that provides a hint to an OpenMP implementation about the desired behavior of waiting threads. Valid values for *policy* are **ACTIVE** (waiting threads consume processor cycles while waiting) and **PASSIVE**.

Tool Activation

Activating an OMPT Tool [4.2]

There are three steps an OpenMP implementation takes to activate a tool. This section explains how the tool and an OpenMP implementation interact to accomplish these tasks.

Step 1. Determine whether to initialize [4.2.2]
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 **ompt_start_tool**.

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.

Step 2. Initializing a first-party tool [4.2.3]
If a tool-provided implementation of **ompt_start_tool** returns a non-**NULL** pointer to an **ompt_start_tool_result_t** structure, the OpenMP implementation will invoke the tool initializer specified in this structure prior to the occurrence of any OpenMP event.

Step 3. Monitoring activity on the host [4.2.4]
To monitor execution of an OpenMP program on the host device, a tool's initializer must register to receive notification of events that occur as an OpenMP program executes. A tool can register callbacks for OpenMP events using the runtime entry point known as **ompt_set_callback**, which has the following possible return codes:

ompt_set_error
ompt_set_never
ompt_set_sometimes
ompt_set_sometimes_paired
ompt_set_always

If the **ompt_set_callback** runtime entry point is called outside a tool's initializer, registration of supported callbacks may fail with a return code of **ompt_set_error**.

All callbacks registered with **ompt_set_callback** or returned by **ompt_get_callback** use the dummy type signature **ompt_callback_t**. While this is a compromise, it is better than providing unique runtime entry points with a precise type signatures to set and get the callback for each unique runtime entry point type signature.

Learn More About OpenMP

OpenMPCon Developer's Conference
Held back-to-back with IWOMP, the annual OpenMPCon conference is organized by and for the OpenMP community to provide both novice and experienced developers tutorials and new insights into using OpenMP and other directive-based APIs.
openmpcon.org

IWOMP International OpenMP Workshop
The annual International Workshop on OpenMP (IWOMP) is dedicated to the promotion and advancement of all aspects of parallel programming with OpenMP, covering issues, trends, recent research ideas, and results related to parallel programming with OpenMP.
iwomp.org

ISC and Supercomputing Conference Series
The annual ISC and SC conferences provide the high-performance computing community with technical programs that makes them yearly must-attend forums. OpenMP has a booth or holds sessions at one or more of these events every year.
supercomputing.org
isc-hpc.com

UK OpenMP Users Conference
The annual UK OpenMP Users Conference provides two days of talks and workshops aimed at furthering collaboration and knowledge sharing among the UK community of expert and novice high-performance computing specialists using the OpenMP API.
ukopenmpusers.co.uk

Copyright © 2018 OpenMP Architecture Review Board.
Permission to copy without fee all or part of this material is granted, provided the OpenMP Architecture Review Board copyright notice and the title of this document appear. Notice is given that copying is by permission of the OpenMP Architecture Review Board. Products or publications based on one or more of the OpenMP specifications must acknowledge the copyright by displaying the following statement: "OpenMP is a trademark of the OpenMP Architecture Review Board. Portions of this product/publication may have been derived from the OpenMP Language Application Program Interface Specification."

OpenMP®