OpenMP 5.0 API Syntax Reference Guide

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.

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 any declarative directive may not appear in Fortran PURE procedures.

### Variant Directives [2.3.4]
A directive that can specify multiple directive variants, one of which may be conditionally selected to replace the metadirective based on the enclosing OpenMP context.

#### Declare Variant [2.3.5]
Declares a specialized variant of a base function and the context in which it is used.

#### Declare Directive Requires [2.4]
Specifies the features that an implementation must provide in order for the code to compile and to execute correctly.

### Parallel Construct

#### Parallel [2.4.5]
Creates a team of OpenMP threads that execute the region.

#### Teams Construct

#### Worksharing Construct

Worksharing loops construct

<table>
<thead>
<tr>
<th>for / do [2.9.2] [2.7.1]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Specifies that the iterations of associated loops will be executed in parallel by threads in the team.</td>
</tr>
</tbody>
</table>

#### Worksharing Loop Construct

<table>
<thead>
<tr>
<th>$SOMP do [clause[, clause]] ...</th>
</tr>
</thead>
<tbody>
<tr>
<td>$SOMP do (clause[, clause]) ...</td>
</tr>
<tr>
<td>$SOMP do (clause[, clause]) ...</td>
</tr>
</tbody>
</table>

### Worksharing Constructs

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.

<table>
<thead>
<tr>
<th>$SOMP sections [clause[, clause]] ...</th>
</tr>
</thead>
<tbody>
<tr>
<td>$SOMP sections [clause] ...</td>
</tr>
<tr>
<td>$SOMP sections [clause] ...</td>
</tr>
</tbody>
</table>

© 2019 OpenMP ARB

OMPO519-01-OMPS
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 an 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 \( chunk\_size / \text{simd\_width} \) \* \text{simd\_width} where \text{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.

```
#pragma omp simd [clause[ [, ]clause] ...] 
```

clause:  
- safelen(length), simdlen(length) 
- linear(list[ : linear-step]) 
- aligned(list[ : alignment]) 
- nonparallel(list) 
- private(list) 
- lastprivate(lastprivate-modifier : list) 
- reduction(reduction-modifier : reduction-identifier : list) 
- collapse(n) 
- dist_schedule(kind, chunk_size) 
- allocate(allocaitor : list)

**do-loops**
```
#pragma omp do simd [clause[ [, ]clause] ...] 
```

clause: Any of the clauses accepted by `simd` or `do`.

**Distribute Parallel Worksharing-Loop**

```
#pragma omp distribute parallel for simd [clause[ [ , ]clause] ...] 
```

clause: Any of the clauses accepted by `simd` or parallel worksharing-loop directives with identical meanings and restrictions.

**Distribute Parallel Worksharing-Loop SIMD**

```
#pragma omp distribute parallel for simd \ { [clause[ [, ]clause] ...] 
```

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

**loop construct**

```
#pragma omp loop [clause[ [, ]clause] ...] 
```

clause:  
- bind(binding) 
- collapse(n) 
- order(concurrent) 
- private(list) 
- lastprivate(list) 
- reduction(reduction-identifier : list)

**loop directive**

```
#pragma omp loop [clause[ [, ]clause] ...] 
```

clause:  
- bind(binding) 
- collapse(n) 
- order(concurrent) 
- private(list) 
- lastprivate(list) 
- reduction(reduction-identifier : list)

**scan directive**

```
#pragma omp scan [clause[ [, ]clause] ...] 
```

clause:  
- reduction( reduction-identifier : list)

**task directive**

```
#pragma omp task [clause[ [, ]clause] ...] 
```

clause:  
- private(list) 
- firstprivate(list) 
- lastprivate(list) 
- collapse(n) 
- in_reduction(list) 
- use_private(list) 
- collapse(0)

**taskloop**

```
#pragma omp taskloop [clause[ [, ]clause] ...] 
```

clause:  
- parallel(parallel-directive) 
- clustered(cluster-directive) 
- nowait

**loop-associative-directive**

```
#pragma omp loop-associative-directive [clause[ [, ]clause] ...] 
```

clause:  
- type(op_event_handle_kind)

**intrinsic directive**

```
#pragma omp intrinsic [clause[ [, ]clause] ...] 
```

clause:  
- scalar-logical-expression

**argument-directive**

```
#pragma omp argument-directive [clause[ [, ]clause] ...] 
```

clause:  
- allocator(list)

**taskloop directive**

```
#pragma omp taskloop [clause[ [, ]clause] ...] 
```

clause:  
- taskloop-directive: taskloop-directive list

**end-taskloop directive**

```
#pragma omp end taskloop [clause[ [, ]clause] ...] 
```

clause:  
- taskloop-directive: taskloop-directive list

**end-taskloop directive**

```
#pragma omp end taskloop [clause[ [, ]clause] ...] 
```

clause:  
- taskloop-directive: taskloop-directive list

**for-directive**

```
#pragma omp for [clause[ [, ]clause] ...] 
```

clause:  
- firstprivate(list) 
- lastprivate(list) 
- collapse(n)

**do-directive**

```
#pragma omp do [clause[ [, ]clause] ...] 
```

clause:  
- private(list)

**distribution-directive**

```
#pragma omp distribution [clause[ [, ]clause] ...] 
```

clause:  
- distribute(safety)

**parallel-directive**

```
#pragma omp parallel [clause[ [, ]clause] ...] 
```

clause:  
- target indictor

**reduce-directive**

```
#pragma omp reduce [clause[ [, ]clause] ...] 
```

clause:  
- reduction-function

**with-directive**

```
#pragma omp with [clause[ [, ]clause] ...] 
```

clause:  
- on (thread-binding)

**do-termination-stmts**

```
#pragma omp do-termination-stmts [clause[ [, ]clause] ...] 
```

clause:  
- unroll

**reduction-identifier**

```
#pragma omp reduction-identifier [clause[ [, ]clause] ...] 
```

clause:  
- reduction-function

**in_reduction**

```
#pragma omp in_reduction [clause[ [, ]clause] ...] 
```

clause:  
- reduction-function

**use_private**

```
#pragma omp use_private [clause[ [, ]clause] ...] 
```

clause:  
- reduction-function

**private**

```
#pragma omp private [clause[ [, ]clause] ...] 
```

clause:  
- reduction-function

**lastprivate**

```
#pragma omp lastprivate [clause[ [, ]clause] ...] 
```

clause:  
- reduction-function

**collapse**

```
#pragma omp collapse [clause[ [, ]clause] ...] 
```

clause:  
- reduction-function

**order**

```
#pragma omp order [clause[ [, ]clause] ...] 
```

clause:  
- reduction-function

**firstprivate**

```
#pragma omp firstprivate [clause[ [, ]clause] ...] 
```

clause:  
- reduction-function

**lastprivate**

```
#pragma omp lastprivate [clause[ [, ]clause] ...] 
```

clause:  
- reduction-function

**default**

```
#pragma omp default [clause[ [, ]clause] ...] 
```

clause:  
- reduction-function

**shared**

```
#pragma omp shared [clause[ [, ]clause] ...] 
```

clause:  
- reduction-function

**firstprivate**

```
#pragma omp firstprivate [clause[ [, ]clause] ...] 
```

clause:  
- reduction-function

**lastprivate**

```
#pragma omp lastprivate [clause[ [, ]clause] ...] 
```

clause:  
- reduction-function

**default**

```
#pragma omp default [clause[ [, ]clause] ...] 
```

clause:  
- reduction-function
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 iterations will also be executed in parallel using OpenMP tasks.

pragma omp taskloop simd [clause [ , [ clause ] ...]]
for-loops

pragma omp taskloop simd [clause [ , [ clause ] ...]]
do-loops

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

taskyield [2.10.4] [2.11.1]
Specifies that the current task can be suspended in favor of execution of a different task.

pragma omp taskyield

Memory management directive

Memory spaces [2.11.1]
Predefined memory spaces [Table 2.8, 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 high capacity.
omp_opt_mem_space | Storage optimized for variables with constant values.
omp_low_lat_mem_space | Storage with low bandwidth.
omp_high_bw_mem_space | Storage with high bandwidth.
omp_large_cap_mem_space | Storage with high capacity.
omp_large_mem_space | Storage with high capacity.

allocate [2.11.3]
Specifies how a set of variables is allocated.

pragma omp allocate [list [ , [ list ] ...]]
or
pragma omp [ alloc directives ]

allocate [list]
clause: Any accepted by the allocate or alloc directives with identical meanings and restrictions.
allocate statement
Clause: allocate (alloctor)
- where alloctor is an expression of:
  C/C++
    C/C++
      type: omp_allocator_handle_t
      kind: omp_allocator_handle_kind

Device directives and construct

target data [2.12.2] [2.10.1]
Creates a device data environment for the extent of the region.

pragma omp target data [clause [ , [ clause ] ...]]
structured-block

pragma omp target data [clause [ , [ clause ] ...]]
structured-block

!$omp target data [clause [ , [ clause ] ...]]
end target data
clause: Any accepted by the target or target directives with identical meanings and restrictions.

allocate [list]
clause: Any accepted by the allocate or alloc directives with identical meanings and restrictions.
allocate statement
Clause: allocate (alloctor)
- where alloctor is an expression of:
  C/C++
    C/C++
      type: omp_allocator_handle_t
      kind: omp_allocator_handle_kind

dependency array alloc

target exit data [2.12.4] [2.10.3]
Unmaps variables from a device data environment.

pragma omp target exit data [clause [ , [ clause ] ...]]
structured-block

pragma omp target exit data [clause [ , [ clause ] ...]]
structured-block

!$omp target exit data
end target data
clause: Any accepted by the target or target directives with identical meanings and restrictions.

dependency array alloc

combined constructs

Parallel Worksharing Loop [2.13.1] [2.11.1]
Specifies a parallel construct containing a worksharing-loop construct with one or more associated loops.

pragma omp parallel for [clause [ , [ clause ] ...]]
for-loop

pragma omp parallel do [clause [ , [ clause ] ...]]
do-loops

pragma omp end parallel do
clause: Any accepted by the parallel or for/do directives, except the nowait clause, with identical meanings and restrictions.

parallel loop [2.13.2]
Shortcut for specifying a parallel construct containing a loop construct with one or more associated loops and no other statements.

pragma omp parallel loop [clause [ , [ clause ] ...]]
for-loops

pragma omp parallel do [clause [ , [ clause ] ...]]
do-loops

pragma omp end parallel do
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 a sections construct and no other statements.

pragma omp parallel sections [clause [ , [ clause ] ...]]
sections

pragma omp parallel sections [clause [ , [ clause ] ...]]
sections

!$omp parallel sections [clause [ , [ clause ] ...]]
for-loops

pragma omp parallel do [clause [ , [ clause ] ...]]
do-loops

pragma omp end parallel do
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 a workshare construct and no other statements.

pragma omp parallel workshare [clause [ , [ clause ] ...]]
workshare

pragma omp parallel do [clause [ , [ clause ] ...]]
do-loops

pragma omp end parallel workshare
clause: Any of the clauses accepted by the parallel directive, with identical meanings and restrictions.
### Directives and Constructs (continued)

#### Parallel Worksharing-Loop SIMD [2.13.3][2.11.14]
Shortcut for specifying a parallel construct containing only one work-sharing-loop SIMD construct.

```
#pragma omp parallel simd [clause [, clause] ...]
```
clause: Any accepted by the parallel or for/do simd directives except for nowait, with identical meanings and restrictions.

#### parallel master [2.13.6]
Shortcut for specifying a parallel construct containing a master construct and no other statements.

```
#pragma omp parallel master [clause [, clause] ...]
```
clause: Any clause used for parallel directive with identical meanings and restrictions.

#### master taskloop [2.11.7]
Shortcut for specifying a master construct containing a taskloop construct and no other statements.

```
#pragma omp master taskloop [clause [, clause] ...]
```
clause: Any clause used for taskloop directive with identical meanings and restrictions.

#### master taskloop simd [2.11.8]
Shortcut for specifying a master construct containing a taskloop simd construct and no other statements.

```
#pragma omp master taskloop simd
{clause [, clause] ...}
```
```
#pragma omp master taskloop simd [clause [, clause] ...]
```
```
#pragma omp end master taskloop
```
clause: Any clause used for taskloop simd directive with identical meanings and restrictions.

#### parallel master taskloop [2.13.9]
Shortcut for specifying a parallel construct containing a master taskloop construct and no other statements.

```
#pragma omp parallel master taskloop
{clause [, clause] ...}
```
```
#pragma omp parallel master taskloop [clause [, clause] ...]
```
```
#pragma 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.

```
#pragma omp parallel master taskloop simd
{clause [, clause] ...}
```
```
#pragma omp parallel master taskloop simd [clause [, clause] ...]
```
```
#pragma 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.

#### Target Parallel Worksharing-Loop SIMD [2.13.17][2.11.6]
Shortcut for specifying a target construct with a parallel work-sharing-loop SIMD construct and no other statements.

```
#pragma omp target parallel for
{clause [, clause] ...}
```
```
#pragma omp target parallel for [clause [, clause] ...]
```
```
#pragma omp end target parallel
```
```
#pragma omp end target parallel do
```
clause: Any accepted by the target or parallel for/do directives, except for copyin, with identical meanings and restrictions.

#### Teams Distribute Parallel Worksharing-Loop SIMD [2.13.13][2.11.14]
Shortcut for specifying a teams construct containing a distribute work-sharing-loop SIMD construct and no other statements.

```
#pragma omp teams distribute parallel for
{clause [, clause] ...}
```
```
#pragma omp teams distribute parallel for [clause [, clause] ...]
```
```
#pragma omp end teams distribute
```
```
#pragma omp end teams distribute simd
```
clause: Any accepted by the teams or parallel distribute directives 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.

```
#pragma omp teams distribute [clause [, clause] ...]
```
```
#pragma omp teams distribute [clause [, clause] ...]
```
```
#pragma omp end teams distribute
```
```
#pragma omp end teams distribute simd
```
clause: Any accepted by the teams or distribute simd directives with identical meanings and restrictions.

#### Target Parallel Worksharing-Loop [2.13.18][2.11.7]
Shortcut for specifying a target construct with a parallel loop work-sharing-loop SIMD construct and no other statements.

```
#pragma omp target parallel loop
{clause [, clause] ...}
```
```
#pragma omp target parallel loop [clause [, clause] ...]
```
```
#pragma omp end target parallel loop
```
```
#pragma omp end target parallel loop simd
```
clause: Any accepted by the target or parallel loop directives with identical meanings and restrictions.

#### target parallel loop [2.13.19]
Shortcut for specifying a target construct containing a parallel loop construct and no other statements.

```
#pragma omp target parallel loop
{clause [, clause] ...}
```
```
#pragma omp target parallel loop [clause [, clause] ...]
```
```
#pragma omp end target parallel loop
```
```
#pragma omp end target parallel loop simd
```
clause: Any accepted by the target or parallel loop simd directives, except for copyin, with identical meanings and restrictions.

#### target simd [2.13.20][2.11.8]
Shortcut for specifying a target construct containing a simd construct and no other statements.

```
#pragma omp target simd
{clause [, clause] ...}
```
```
#pragma omp target simd [clause [, clause] ...]
```
```
#pragma omp end target simd
```
```
#pragma 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]
Shortcut for specifying a target construct containing a teams construct and no other statements.

```
#pragma omp target teams
{clause [, clause] ...}
```
```
#pragma omp target teams [clause [, clause] ...]
```
```
#pragma omp end target teams
```
```
#pragma 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]
Shortcut for specifying a target construct containing a teams distribute construct and no other statements.

```
#pragma omp target teams distribute
{clause [, clause] ...}
```
```
#pragma omp target teams distribute [clause [, clause] ...]
```
```
#pragma omp end target teams distribute
```
```
#pragma omp end target teams distribute
```
clause: Any accepted by the target or teams distribute directives with identical meanings and restrictions.
Directives and Constructs (continued)

target teams distribute simd [2.13.23] [2.11.13]
Shortcut for specifying a target construct containing a
target teams distribute simd construct and no other statements.

!$omp target teams distribute simd \[
[clause [, clause] ...]
for-loops

!$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
tasks loop construct and no other statements.

!$omp target teams loop \[
[clause [, clause] ...]
for-loops

!$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
tasks distribute parallel worksharing-loop construct and
no other statements.

!$omp target teams distribute parallel for \[
[clause [, clause] ...]
for-loops

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

!$omp end target teams distribute parallel do/
clause: Any clause used for target or teams distribute parallel for/do 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
tasks distribute parallel worksharing-loop SIMD construct and
no other statements.

!$omp target teams distribute parallel for simd \[
[clause [, clause] ...]
for-loops

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

!$omp end target teams distribute parallel do simd/
clause: Any clause used for target or teams distribute parallel for/simd 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.

!$omp master \[
structured-block

!$omp end master

Synchronization constructs

critical [2.17.1] [2.12.2]
Restricts execution of the associated structured block to
a single thread at a time.

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

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

!$omp end critical ([name])

hint-expression: 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 sync_hind kind and a value that is a
valid synchronization hint.

barrier [2.17.2] [2.13.3]
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.

!$omp barrier

taskwait [2.17.5] [2.13.4]
Specifies a wait on the completion of child tasks of the
current task.

!$omp taskwait \[clause \[
[, ]clause ...\]
clause: depend ([depend-modifier, dependence-type : 
locator-list])

taskgroup [2.17.6] [2.13.5]
Specifies a region which a task cannot leave until all its
descendant tasks generated inside the dynamic scope of the
region have completed

!$omp taskgroup \[clause \[
[, ]clause ...\]
structured-block

!$omp end taskgroup

taskgroup: 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:

!$omp atomic ([clause[ , clause] ...] [, clause[ , clause] ...])
expression-stmt

!$omp atomic ([clause[ , clause] ...] [, clause[ , clause] ...])
expression-stmt

!$omp atomic ([clause[ , clause] ...] [, clause[ , clause] ...])
expression-stmt

!$omp atomic ([clause[ , clause] ...] [, clause[ , clause] ...])
expression-stmt

!$omp atomic ([clause[ , clause] ...] [, clause[ , clause] ...])
expression-stmt

!$omp atomic ([clause[ , clause] ...] [, clause[ , clause] ...])
expression-stmt

!$omp atomic ([clause[ , clause] ...] [, clause[ , clause] ...])
expression-stmt

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.

!$omp flush memory-order-clause : [list]

memory-order-clause: seq, cst, acq_rel, release, acquire
relaxed

!$omp flush memory-order-clause : [list]

memory-order-clause: seq, cst, acq_rel, release, acquire
relaxed

ordered [2.17.9] [2.13.8]
Specifies a structured block that is to be executed in
loop iteration order in a parallelized loop, or it specifies
cross iteration dependencies in a doacross loop nest.

!$omp ordered [clause[ , clause] ...]
structured-block

- or -

!$omp ordered [clause[ , clause] ...]

ordered: clause (for the first form): threads or simd
ordered: clause (for the second form):
depend (source) or depend (sink : vec)
### Directives and Constructs (continued)

**Depobj** [2.17.10.1]

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

```
#pragma omp depobj (depoj) clause
|omp depobj (depoj) clause
```

**C/C++**

Clauses:
- `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.

```
#pragma omp cancel construct-type-clause [ , ] if-clause
|omp cancel construct-type-clause [ , ] if-clause
```

**C/C++**

For
- `construct-type-clause`: parallel, sections, taskgroup, for
- `if-clause`: if ([ cancel ] : scalar-expression)

### Data environment directives

**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.

```
#pragma omp threadprivate (list)
|omp threadprivate (list)
```

**C/C++**

- `list`: A comma-separated list of file-scope, namespace-scope, or static block-scope variables that do not have incomplete types
- `for` construct:
  - `construct-type-clause`: parallel, sections, taskgroup, do
  - `if-clause`: if ([ cancel ] : scalar-logical-expression)

### 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 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`.

```
void omp_set_num_threads (int num_threads);
```

**Omp_get_num_threads** [3.2.2] [3.2.2]

Returns the number of threads in the current thread. 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.

```
int omp_get_num_threads (void);
```

**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.

```
int omp_get_max_threads (void);
```

**Omp_get_thread_num** [3.2.4] [3.2.4]

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

```
int omp_get_thread_num (void);
```

**Omp_get_num_procs** [3.2.5] [3.2.5]

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

```
int omp_get_num_procs (void);
```

**Omp_get_in_parallel** [3.2.6] [3.2.6]

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

```
int omp_get_in_parallel (void);
```

**Omp_get_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 `dynt-var` ICV.

```
void omp_set_dynamic (int dynamic_threads);
```

**Omp_get_dynamic** [3.2.8] [3.2.8]

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

```
int omp_get_dynamic (void);
```

**Omp_get_cancellation** [3.2.9] [3.2.9]

Returns the value of the cancellation point variable that can be used in a reduction clause.

```
int omp_get_cancellation (void);
```

**Omp_get_nested** [3.2.10] [3.2.10]

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

```
void omp_set_nested (int 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.

```
int omp_get_nested (void);
```

**Omp_get_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.

```
void omp_set_schedule (omp_sched_t kind, int chunk_size);
```

### Declare mapper

**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.

```
#pragma omp declare mapper (mapper-identifier : type var) [ class ] ...
```

**C/C++**

- `mapper-identifier`: A base-language identifier or default type: A valid type in scope
- `var`: A valid variable in scope
- `class`: A comma-separated list of class expressions

### Declaration of reduction

**Declare reduction** [2.19.5.7] [2.16]

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

```
#pragma omp declare reduction ( \
reduction-identifier : typename-list : combiner ) [ initializer-clause]
```

**C/C++**

- `typename-list`: A list of type names
- `combiner`: An expression
- `initializer-clause`: A base language identifier (for C), or an id-expression (for C++), or one of the following operators: `+, *`, `<`, `>`, `<=`, `>=`, `==`, `!=`, `&`, `|`, `^`, `&&`, `||`

### Dependence expression

**Dependence expression** [2.18.1]

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

```
#pragma omp declare reduction ( \
dependence-type : depobj) [ init-value ] ...
```

**C/C++**

- `dependence-type`: A base-language identifier or an `omp_priv` modifier
- `depobj`: A comma-separated list of file-scope, namespace-scope, or static block-scope variables that do not have incomplete types
- `for` construct:
  - `construct-type-clause`: parallel, sections, taskgroup, do
  - `if-clause`: if ([ cancel ] : scalar-logical-expression)

### Alloc and destroy dependence

**Alloc and destroy dependence** [2.15.2]

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

```
void omp_set_dynamic (int dynamic_threads);
```
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
void omp_get_schedule (omp_sched_t *kind, int *chunk_size);
```

`omp_get_partition_num_places`
Returns numerical identifiers of the processors available to the execution environment in the specified place.
```c
void omp_get_partition_num_places (place_nums);
```

`omp_get_partition_num_places`
Returns the value of places in the place-partition-var ICV of the innermost implicit task.
```c
void omp_get_partition_place_nums (place_nums);
```

`omp_get_max_active_levels`
Returns the number of active levels of parallelism supported.
```c
int omp_get_max_active_levels (void);
```

`omp_get_max_active_levels`
Returns the number of active levels of parallelism supported.
```c
int omp_get_max_active_levels (void);
```

`omp_get_partition_place_nums`
Returns numerical identifiers of the processors available to the execution environment in the specified place.
```c
void omp_get_partition_place_nums (place_nums);
```

`omp_set_affinity_format`
Sets the affinity format to be used on the device by setting the value of the affinity-format-var ICV.
```c
void omp_set_affinity_format (const char *format);
```

`omp_get_schedule`
Returns the value of the run-sched-var ICV, which is the schedule applied when runtime schedule is used.
```c
int omp_get_schedule (void);
```

`omp_in_final`
Returns true if the routine is executed in a final task region; otherwise, it returns false.
```c
int omp_in_final (void);
```

`omp_get_places`
Returns the number of processors available to the execution device.
```c
int omp_get_num_places (void);
```

`omp_get_affinity_format`
Returns the value of the affinity-format-var ICV on the device.
```c
size_t omp_get_affinity_format (char *buffer, size_t size);
```

`omp_get_places`
Returns the number of processors available to the execution device.
```c
int omp_get_num_places (void);
```

`omp_set_default_device`
Assigns the value of the default-device-var ICV, which determines default target device.
```c
void omp_set_default_device (int device_num);
```

`omp_get_places`
Returns the number of processors available to the execution device in the specified place.
```c
int omp_get_num_places (int place_num);
```

`omp_get_schedule`
Returns the value of the run-sched-var ICV, which is the schedule applied when runtime schedule is used.
```c
int omp_get_schedule (void);
```

`omp_get_places`
Returns the number of places in the place-partition-var ICV of the innermost implicit task.
```c
int omp_get_partition_num_places (void);
```

`omp_set_default_device`
Assigns the value of the default-device-var ICV, which determines default target device.
```c
void omp_set_default_device (int device_num);
```

`omp_get_places`
Returns the number of places in the place-partition-var ICV of the innermost implicit task.
```c
int omp_get_partition_num_places (void);
```

`omp_set_default_device`
Assigns the value of the default-device-var ICV, which determines default target device.
```c
void omp_set_default_device (int device_num);
```

`omp_get_places`
Returns the number of places in the place-partition-var ICV of the innermost implicit task.
```c
int omp_get_partition_num_places (void);
```

`omp_set_default_device`
Assigns the value of the default-device-var ICV, which determines default target device.
```c
void omp_set_default_device (int device_num);
```
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 a teams region.

```c
int omp_get_num_teams (void);
```

**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
int omp_get_team_num (void);
```

**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
int omp_is_initial_device (void);
```

**omp_get_initial_device [3.2.41] [3.2.35]**
Returns a device number representing the host device.

```c
int omp_get_initial_device (void);
```

**omp_get_max_task_priority [3.2.42] [3.2.36]**
Returns the maximum value that can be specified in the priority clause.

```c
int omp_get_max_task_priority (void);
```

**omp_pause_resource [3.2.43]**
**omp_pause_resource_all [3.2.44]**
Allows the runtime to relinquish resources used by OpenMP on the specified device. Valid kind values include `omp_pause_soft` and `omp_pause_hard`.

```c
int omp_pause_resource [kind, int device_num];
int omp_pause_resource_all [kind, int device_num];
```

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

**Initialize lock [3.3.1] [3.3.1]**
Initializes an OpenMP lock.

```c
void omp_init_lock (omp_lock_t *lock);
void omp_init_nest_lock (omp_nest_lock_t *lock);
```

**Initialize lock with hint [3.3.2] [3.3.2]**
Initializes an OpenMP lock with a hint.

```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);
```

**Set lock [3.3.3] [3.3.3]**
Ensures that the OpenMP lock is initialized.

```c
void omp_set_lock (ompplock_t *lock);
void omp_set_nest_lock (omp_nest_lock_t *lock);
```

**Set lock [3.3.3] [3.3.3]**
Sets an OpenMP lock. The calling task region is suspended until the lock is set.

```c
void omp_set_lock (omp_lock_t *lock);
void omp_set_nest_lock (omp_nest_lock_t *lock);
```

**Destroy lock [3.3.3] [3.3.3]**
Unsets an OpenMP lock.

```c
void omp_destroy_lock (omp_lock_t *lock);
void omp_destroy_nest_lock (omp_nest_lock_t *lock);
```

**Unset lock [3.3.5] [3.3.5]**
Unsets an OpenMP lock.

```c
void omp_unset_lock (omp_lock_t *lock);
void omp_unset_nest_lock (omp_nest_lock_t *lock);
```

**Test lock [3.3.6] [3.3.6]**
Attempts to set an OpenMP lock but do not suspend execution of the task executing the routine.

```c
int omp_test_lock (omp_lock_t *lock);
int omp_test_nest_lock (omp_nest_lock_t *lock);
```

**Testing 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
double omp_get_wtime (void);
```

**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
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
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
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
int omp_target_memcpy (void *dst, const void *src, size_t size, int dst_device_num, int src_device_num, int src_device_offset, int dst_device_offset);
```

**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
int omp_target_memcpy_rect (void *dst, const void *src, size_t src_device_offset, size_t dst_device_offset, size_t device_num, size_t size, int src_device_num, int src_device_dim, int dst_device_dim);
```

**omp_target_associate_ptr [3.6.6] [3.5.6]**
Maps storage to which a device pointer points to storage to which a host pointer points. The device pointer may be the result of a call to `omp_target_alloc` and may have been obtained from implementation-defined runtime routines.

```c
int omp_target_associate_ptr (void *host_ptr, const void *device_ptr, int size, int device_offset, int device_num);
```

**omp_target_launch [3.6.7] [3.5.7]**
Removes the association between a host pointer and a device address on a given device.

```c
int omp_target_launch (void *ptr, int device_num);
```
Runtime Library Routines (continued)

Memory management routines

Memory Management Types [3.7.1]
The `omp_alloctrait_t` struct in C/C++ and `omp_alloctrait` type in Fortran define members named `key` and `value`, with these types and values:

```c
enum omp_alloctrait_key_t { // C/C++
    integer omp_alloctrait_key_kind
}
```

```fortran
module omp_alloctrait_val_kind
    integer omp_alloctrait_value_t
endmodule
```

The `omp_alloctrait_t` struct has the following fields:

- `member (kind=omp_control_tool_kind)`
- `region`
- `void [depend-modifier,]`
- `list [allocator:]`
- `linear-list [::]`

These members are used to define the types and values associated with the memory management routines.

Allocate Clause [2.11.4]

```c
allocate (allocators: list)
```

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

- `allocators: list`
  - `allocator`
    - `C/C++`: Expression of type `omp_allocator_handle_t`
    - `Fortran`: 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]

- `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.

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` must be source.

- `depend (dependence-type : vec)`
  - `dependence-type` must be sink and vec is an iteration vector with form: $x_1 \pm d_1, x_2 \pm d_2, \ldots, x_n \pm d_n$

```
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.

- `implicit-behavior`: alloc, to, from, tofrom, firstprivate, none, default
- `variable-category`: C/C++
  - scalar, aggregate, pointer
- `variable-category`: Fortran
  - scalar, aggregate, pointer, allocatable

```
End``
**Clauses (continued)**

**Depend (continued)**

- `out` and `inout`: The generated task will be a depend task of all previously generated sibling tasks that reference at least one of the list items in an `in`, `out`, `mutxinsout`, or `inout dependence-type` list.

- `mutxinsoutset`: 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 depend 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 mutxinsoutset dependence-type on a construct from which a sibling task was previously generated, then the sibling tasks will be mutually exclusive tasks.

- `depoj`: The task dependencies are derived from the depend clause specified in the depoj constructs that initialized dependencies represented by the depend objects specified on in the depend clause as if the depend clauses of the depoj 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]) Fortran

**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`)

**Ordered Clause [2.9.2]**

Ordered (n) Indicates the loops or how many loops to associate with a construct.

**Reduction Clauses [2.19.5] [2.15.3.6]**

- `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]**

Also see Data Sharing Attribute Clauses and If Clause in this guide.

- `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.

- `collapse` (n)
  A constant positive integer expression that specifies how many loops are associated with the construct. (Not used in declare simd.)

- `inbranch`
  Specifies that the function will always be called from inside a conditional statement of a SIMD loop. (Not used in simd.)

- `nontemporal`
  Specifies that accesses to the storage locations to which the list items refer have low temporal locality across the iterations in which those storage locations are accessed.

- `notinbranch`
  Specifies that the function will never be called from inside a conditional statement of a SIMD loop. (Not used in simd.)

- `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. (Not used in declare simd.)

- `simdlen` (length)
  A constant positive integer expression that specifies the preferred number of iterations to be executed concurrently.

- `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. (Not used in simd.)

**Tasking Clauses [2.10] [2.9]**

- `affinity` (affinity-selector; locator-list)
  A hint to execute closely to the location of the list items. (Not used in taskloop.)

- `allocate` (locator-list)
  See Allocate Clause, page 9 of this guide.

- `collapse` (n)
  See SIMD Clauses on this page. (Not used in task.)

- `default` (shared | none) C/C++
  default (private | firstprivate | shared | none) Fortran
  See Data Sharing Attribute Clauses, page 9 of this guide.

- `depend` (depoj; dependence-type: locator-list)
  See Depend Clause, page 9 of this guide. (Not used in taskloop.)

- `detach` (list)
  Causes an implicit reference to the variable list in all enclosing constructs. (Not used in taskloop.)

- `final` (scalar-expression) C/C++
  final (scalar-logical-expression) Fortran
  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` (gran-size)
  Causes the number of logical loop iterations assigned to each created task to be greater than or equal to the minimum of the value of the grain-size expression and the number of logical loop iterations, but less than twice the value of the grain-size expression. (Not used in task.)

- `in_reduction` (reduction-identifier: list)
  See Reduction Clauses on this page.

- `lastprivate` (list)
  See Data Sharing Attribute Clauses, page 9 of this guide. (Not used in task.)

- `mergeable`
  Specifies that the generated task is a mergeable task.

- `nogroup`
  Prevents an implicit taskgroup region to be created. (Not used in task.)

- `num_tasks` (num-tasks)
  Create as many tasks as the minimum of the num-tasks expression and the number of logical loop iterations. (Not used in task.)

- `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.

- `route` (reduction-modifier; reduction-identifier: list)
  See Reduction Clauses on this page. (Not used in task.)

- `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 [2.1.6]**

- `identifiers` [2.1.6]
  Identifiers that expand to multiple values in the clause on which they appear.

- `iterator` (iterators-definition)
  Iterators-definition:
  - `iterator-specifier`: [ iterator-type ] iterator-identifier = range-specification
  - `iterators-specifier`: [ iterator-type ] iterator-identifier = range-specification

- `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. Fortran
### 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>
<thead>
<tr>
<th>ICV</th>
<th>Environment variable</th>
<th>Initial value</th>
<th>Ways to modify value</th>
<th>Ways to retrieve value</th>
<th>Env. Var. Ref.</th>
</tr>
</thead>
<tbody>
<tr>
<td>dyn-var</td>
<td>OMP_DYNAMIC</td>
<td>Implemented, defined if the implementation supports dynamic adjustment of the number of threads; otherwise, the initial value is false.</td>
<td>omp_set_dynamic()</td>
<td>omp_get_dynamic()</td>
<td>[6.3] [4.3]</td>
</tr>
<tr>
<td>nest-var</td>
<td>OMP_NESTED</td>
<td>Implemented defined.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>num_threads-var</td>
<td>OMP_NUM_THREADS</td>
<td>Implemented defined list.</td>
<td>omp_set_num_threads()</td>
<td>omp_get_max_threads()</td>
<td>[6.2] [4.2]</td>
</tr>
<tr>
<td>run-sched-var</td>
<td>OMP_SCHEDULE</td>
<td>Implemented defined.</td>
<td>omp_set_schedule()</td>
<td>omp_get_schedule()</td>
<td>[6.1] [4.1]</td>
</tr>
<tr>
<td>def-sched-var</td>
<td>(none)</td>
<td>Implemented defined.</td>
<td>(none)</td>
<td>(none)</td>
<td></td>
</tr>
<tr>
<td>bind-var</td>
<td>OMP_PROC_BIND</td>
<td>Implemented defined list.</td>
<td>(none)</td>
<td>omp_get_proc_bind()</td>
<td>[6.4] [4.4]</td>
</tr>
<tr>
<td>size-var</td>
<td>OMP_STACKSIZE</td>
<td>Implemented defined.</td>
<td>(none)</td>
<td>(none)</td>
<td>[6.7] [4.7]</td>
</tr>
<tr>
<td>wait-policy-var</td>
<td>OMP_WAIT_POLICY</td>
<td>Implemented defined.</td>
<td>(none)</td>
<td>(none)</td>
<td>[6.7] [4.8]</td>
</tr>
<tr>
<td>thread-limit-var</td>
<td>OMP_THREAD_LIMIT</td>
<td>Implemented defined.</td>
<td>thread_limit clause</td>
<td>omp_get_thread_limit()</td>
<td>[6.10] [6.10]</td>
</tr>
<tr>
<td>max-active-levels-var</td>
<td>OMP_MAX_ACTIVE_LEVELS, OMP_NESTED</td>
<td>The number of levels of parallelism that the implementation supports.</td>
<td>omp_set_max_active_levels()</td>
<td>omp_get_max_active_levels()</td>
<td>[6.8] [4.9]</td>
</tr>
<tr>
<td>active-levels-var</td>
<td>(none)</td>
<td>zero</td>
<td>(none)</td>
<td>omp_get_active_level()</td>
<td></td>
</tr>
<tr>
<td>levels-var</td>
<td>(none)</td>
<td>zero</td>
<td>(none)</td>
<td>omp_get_level()</td>
<td></td>
</tr>
<tr>
<td>place-partition-var</td>
<td>OMP_PLACES</td>
<td>Implemented defined.</td>
<td>(none)</td>
<td>omp_get_partition_num_places()</td>
<td>[6.5] [4.5]</td>
</tr>
<tr>
<td>cancel-var</td>
<td>OMP_CANCELLATION</td>
<td>false</td>
<td>(none)</td>
<td>omp_get_cancellation()</td>
<td>[6.11] [4.11]</td>
</tr>
<tr>
<td>display-affinity-var</td>
<td>OMP_DISPLAY_AFFINITY</td>
<td>Implemented defined.</td>
<td>(none)</td>
<td>(none)</td>
<td>[6.13]</td>
</tr>
<tr>
<td>default-device-var</td>
<td>OMP_DEFAULT_DEVICE</td>
<td>Implemented defined.</td>
<td>omp_set_default_device()</td>
<td>omp_get_default_device()</td>
<td>[6.15] [4.13]</td>
</tr>
<tr>
<td>target-offload-var</td>
<td>OMP_TARGET_OFFLOAD</td>
<td>DEFAULT</td>
<td>(none)</td>
<td>(none)</td>
<td>[6.17]</td>
</tr>
<tr>
<td>target-offload-var</td>
<td>(none)</td>
<td>zero</td>
<td>(none)</td>
<td>omp_get_max_task_priority()</td>
<td>[6.16] [4.14]</td>
</tr>
<tr>
<td>tool-var</td>
<td>OMP_TOOL</td>
<td>enabled</td>
<td>(none)</td>
<td>(none)</td>
<td>[6.18]</td>
</tr>
<tr>
<td>tool-libraries-var</td>
<td>OMP_TOOL_LIBRARIES</td>
<td>empty string</td>
<td>(none)</td>
<td>(none)</td>
<td>[6.19]</td>
</tr>
<tr>
<td>debug-var</td>
<td>OMP_DEBUG</td>
<td>disabled</td>
<td>(none)</td>
<td>(none)</td>
<td>[6.20]</td>
</tr>
<tr>
<td>def-allocator-var</td>
<td>OMP_ALLOCATOR</td>
<td>Implementation defined.</td>
<td>omp_set_default_allocator()</td>
<td>omp_get_default_allocator()</td>
<td>[6.21]</td>
</tr>
</tbody>
</table>

### 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. arg is a case-insensitive, predefined allocator (for details, see Table 2.10):
  - omp_default_mem_alloc
  - omp_large_cap_mem_alloc
  - omp_const_mem_alloc
  - omp_high_bw_mem_alloc
  - omp_thread_mem_alloc
  - omp_group_mem_alloc
  - omp_low_lat_mem_alloc
  - omp_thread_mem_alloc
  - omp_pthread_mem_alloc
  - omp_multithread Allocates memory to the running thread, or to another thread if specified.

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.

<table>
<thead>
<tr>
<th>Allocate trait</th>
<th>Allowed values (default)</th>
</tr>
</thead>
<tbody>
<tr>
<td>sync_hint</td>
<td>contended, uncontended, serialized, private</td>
</tr>
<tr>
<td>alignment</td>
<td>1 byte: Positive integer value that is a power of 2</td>
</tr>
<tr>
<td>access</td>
<td>all, groupop, pthread, thread</td>
</tr>
<tr>
<td>pool_size</td>
<td>Positive integer value (default is implementation defined)</td>
</tr>
<tr>
<td>fallback</td>
<td>default_mem_fb, null_fb, abort_fb, allocator_fb</td>
</tr>
<tr>
<td>fb_data</td>
<td>An allocator handle (No default)</td>
</tr>
<tr>
<td>pinned</td>
<td>true, false</td>
</tr>
<tr>
<td>partition</td>
<td>environment, nearest, blocked, interleaved</td>
</tr>
</tbody>
</table>

- **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 format 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: %([I0] ) size ] type, where the field type may be either the short or long names listed below (Table 6.2).

<table>
<thead>
<tr>
<th>Specifier</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>t</td>
<td>team_num</td>
</tr>
<tr>
<td>T</td>
<td>num_teams</td>
</tr>
<tr>
<td>N</td>
<td>num_threads</td>
</tr>
<tr>
<td>L</td>
<td>nesting_level</td>
</tr>
<tr>
<td>P</td>
<td>process_id</td>
</tr>
<tr>
<td>H</td>
<td>host</td>
</tr>
<tr>
<td>i</td>
<td>native_thread_id</td>
</tr>
</tbody>
</table>

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

- **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, may be TRUE or FALSE. If TRUE, the implementation may dynamically adjust the number of threads to use for executing parallel regions.
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 [6.5][4.6]
  Controls nested parallelism with max-active-levels-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 [6.8][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. B is bytes, K is kilobytes, M is megabytes, and G is gigabytes. If unit is not specified, size is measured in K.

**OMP_TARGET_OFFLOAD** arg [6.17]
Sets the initial value of the target-offload-var ICV. arg 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_TOOLS** 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. library-list is a 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.

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
- ompt_set_impossible

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 experts and novice high-performance computing specialists using the OpenMP API.

ukopenmpusers.co.uk