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 processors, 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.

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.

### Directives and Constructs

**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.

```plaintext
!$omp metadirective [clause [ [, clause] ... ]]
```

**declare variant [2.3.5]**

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

```plaintext
!$omp declare variant [clause [ [, clause] ... ]]
```

**requires directive**

**requires [2.4]**

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

```plaintext
!$omp requires [clause [ [, clause] ... ]]
```

### parallel construct

**parallel [2.6] [2.5]**

Creates a team of OpenMP threads that execute the region.

```plaintext
!$omp parallel [clause [ [, clause] ... ]]
```

### single construct

**single [2.8.2] [2.7.3]**

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

```plaintext
!$omp single [clause [ [, clause] ... ]]
```

### workshare construct

**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.

```plaintext
!$omp workshare [structured-block]
```

### 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.

```plaintext
!$omp for [clause [ [, clause] ... ]]
```

### Worksharable 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.

```plaintext
!$omp sections [clause [ [, clause] ... ]]
```

<table>
<thead>
<tr>
<th>Clause</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>reverse_offload</td>
<td>unified_address</td>
</tr>
<tr>
<td>unified_shared_memory</td>
<td>atomic_default_mem_order(seq_cst</td>
</tr>
<tr>
<td>dynamic_allocators</td>
<td></td>
</tr>
</tbody>
</table>

© 2018 OpenMP ARB

OpenMP® is a registered trademark of the OpenMP ARB. OpenMP API 5.0 Page 1

OMP1118-02-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 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 \( \text{chunk_size}_\text{new} = \text{chunk_size}_\text{old} \times \text{simd_width} \) 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] ... 
  
  P志omp simd [clause] ... do-loops 
  
  P志omp simd [clause] ... do-loops end 
  
  P志omp end simd 
  
  clause:  
  - safelen (length), simdlen (length) 
  - linear (list), affine (line-step) 
  - aligned (list), alignment() 
  - nontemporal (list) 
  - private (list) 
  - lastprivate (list), private (list) 
  - collapse (n), lastprivate (list), collapse (n) 
  - dist_schedule (kind, chunk_size) 
  - allocate (allocator : list) 
  ```

**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.

```
#pragma omp simd [clause] ... for-loops 

#pragma omp simd [clause] ... do-loops 

#pragma omp simd [clause] ... do-loops end 

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

**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.

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

#pragma omp simd declare [proc-name] [clause] ... 
```

**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.

  ```
  #pragma omp loop [clause] ... 
  
  #pragma omp loop [clause] ... do-loops 
  
  #pragma omp loop [clause] ... do-loops end 
  
  #pragma omp loop [clause] ... do-loops end 
  ```

**scan directive**

- **scan** \([2.9.6]\)
  Specifies that scan computations update the list items on each iteration.

  ```
  #pragma omp scan [clause] ... 
  
  #pragma omp scan [clause] ... do-loops 
  
  #pragma omp scan [clause] ... do-loops end 
  
  #pragma omp scan [clause] ... do-loops end 
  ```
Directives and Constructs (continued)

**taskloop** [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
#pragma omp taskloop simd (clause) ... [for-loops]
```

**taskyield** [2.10.4] [2.11.2]

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

```c
#pragma omp taskyield
```

**allocate** [2.11.1]

Specifies how a set of variables is allocated.

```c
#pragma omp allocate (list) (clause)
```

**Memory management directive**

**Memory spaces** [2.11.1]

Predefined memory spaces [Table 2.7, below] represent storage resources for storage and retrieval of variables.

**Device directives and construct**

**target data** [2.12.2] [2.10.1]

Creates a device data environment for extent of the region.

```c
#pragma omp target data clause ( ] [ , ] clause ... ) [structured-block]
```

**target enter data** [2.12.3] [2.10.2]

Maps variables to a device data environment.

```c
#pragma omp target enter data clause ( ] [ , ] clause ... )
```

**target exit data** [2.12.5] [2.10.4]

Unmaps variables from a device data environment.

```c
#pragma omp target exit data clause ( ] [ , ] clause ... )
```

**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
#pragma omp target update clause ( ] [ , ] clause ... )
```

**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 and no other statements.

```c
#pragma omp parallel loop ( ] [ , ] clause ... ) [for-loops]
```

**parallel sections** [2.13.3] [2.11.2]

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

```c
#pragma omp parallel sections ( ] [ , ] clause ... )
```

**parallel workshare** [2.13.4] [2.11.3]

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

```c
#pragma omp parallel workshare ( ] [ , ] clause ... )
```
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.

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

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 taskloop and no other statements.

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

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

```
#pragma omp parallel master do simd [clause[, [clause]] ...]
```

clause: Any clause used for master or parallel 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.

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

```
#pragma omp master taskloop do simd [clause[, [clause]] ...]
```

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

clause: Any clause used for master or taskloop simd 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.

```
#pragma omp master taskloop simd \ [clause[, [clause]] ...]
```

```
#pragma omp master taskloop simd do simd [clause[, [clause]] ...]
```

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

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.

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

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

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

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 do simd [clause[, [clause]] ...]
```

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

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 do-loops construct and no other statements.

```
#pragma omp teams distribute [clause[, [clause]] ...]
```

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

```
#pragma omp teams distribute do simd [clause[, [clause]] ...]
```

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 do-loops construct and no other statements.

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

```
#pragma omp teams distribute parallel do [clause[, [clause]] ...]
```

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

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.

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

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

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

clause: Any accepted by teams or distribute parallel do/simd directives with identical meanings and restrictions.

Target Parallel Worksharing-Loop SIMD [2.13.17] [2.11.8]
Shortcut for specifying a target construct with a parallel worksharing-loop SIMD construct and no other statements.

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

```
#pragma omp target parallel do simd [clause[, [clause]] ...]
```

```
#pragma omp target parallel do simd [clause[, [clause]] ...]
```

clause: Any accepted by the target or parallel for/do simd directives, except for copyin, with identical meanings and restrictions.

Target Parallel Worksharing-Loop [2.13.18] [2.11.9]
Shortcut for specifying a target construct with a parallel worksharing-loop SIMD construct and no other statements.

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

```
#pragma omp target parallel do simd [clause[, [clause]] ...]
```

```
#pragma omp target parallel do simd [clause[, [clause]] ...]
```

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.

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

```
#pragma omp target parallel loop do [clause[, [clause]] ...]
```

```
#pragma omp target parallel loop do [clause[, [clause]] ...]
```

clause: Any accepted by the target 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.

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

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

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

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.

```
#pragma omp target teams [clause[, [clause]] ...]
```

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

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

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.

```
#pragma omp target teams distribute [clause[, [clause]] ...]
```

```
#pragma omp target teams distribute do [clause[, [clause]] ...]
```

```
#pragma omp target teams distribute do [clause[, [clause]] ...]
```

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

- `#pragma omp target teams distribute simd \  
  [clause[, clause]...]  
  [for-loops]`
- `#pragma omp target teams distribute simd  
  [clause[, clause]...]  
  do-loops`
- `#pragma omp target teams distribute simd  
  [clause[, clause]...]  
  [for-loop end target teams distribute simd]`

**Target Teams Loop** [2.13.24]  
Shortcut for specifying a target construct containing a teams loop construct and no other statements.

- `#pragma omp target teams loop \  
  [clause[, clause]...]  
  [for-loops]`
- `#pragma omp target teams loop  
  [clause[, clause]...]  
  do-loops`
- `#pragma omp target teams loop  
  [clause[, clause]...]  
  [for-loop end target teams loop]`

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

- `#pragma omp target teams distribute parallel \  
  [clause[, clause]...]  
  [for-loops]`
- `#pragma omp target teams distribute parallel  
  [clause[, clause]...]  
  do-loops`
- `#pragma omp target teams distribute parallel  
  [clause[, clause]...]  
  [for-loop end target teams distribute parallel]`

**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.

- `#pragma omp target teams distribute parallel for \  
  [clause[, clause]...]  
  [for-loops]`
- `#pragma omp target teams distribute parallel for \  
  [clause[, clause]...]  
  do-loops`
- `#pragma omp target teams distribute parallel for \  
  [clause[, clause]...]  
  [for-loop end target teams distribute parallel for]`

**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.

- `#pragma omp target teams distribute parallel for simd \  
  [clause[, clause]...]  
  [for-loops]`
- `#pragma omp target teams distribute parallel do & \  
  [clause[, clause]...]  
  do-loops`
- `#pragma omp target teams distribute parallel do simd & \  
  [clause[, clause]...]  
  do-loops`

**master construct** [2.13.1]  
Specifies a structured block that is executed by the master thread of the team.

- `#pragma omp master \  
  structured-block`
- `#pragma omp master \  
  structured-block`
- `#pragma omp master \  
  do-loops`

**atomic** [2.13.7] [2.13.6]  
Ensures a specific storage location is accessed atomically. May take one of the following seven forms:

- `#pragma omp atomic [clause[, clause]...] \  
  structured-block`
- `#pragma omp atomic [clause[, clause]...] \  
  expression-stmt`
- `#pragma omp atomic [clause[, clause]...] \  
  capture [clause[, clause]...] \  
  structured-block`
- `#pragma omp atomic [clause[, clause]...] \  
  capture [clause[, clause]...] \  
  expression-stmt`
- `#pragma omp atomic [clause[, clause]...] \  
  capture [clause[, clause]...] \  
  update & [clause[, clause]...] \  
  structured-block`
- `#pragma omp atomic [clause[, clause]...] \  
  capture [clause[, clause]...] \  
  update & [clause[, clause]...] \  
  expression-stmt`
- `#pragma omp atomic [clause[, clause]...] \  
  capture [clause[, clause]...] \  
  write & [clause[, clause]...] \  
  update-stmt`

**Synchronization constructs**

**synchronization directives** [2.13.1.2]  
Restricts execution of the associated structured block to a single thread at a time.

- `#pragma omp critical [name] [hint-expression] \  
  structured-block`
- `#pragma omp critical [name] [hint-expression] \  
  expression-stmt`
- `#pragma omp critical [name] [hint-expression] \  
  write-stmt`
- `#pragma omp critical [name] [hint-expression] \  
  update-stmt`
- `#pragma omp critical [name] [hint-expression] \  
  capture-statement`
- `#pragma omp critical [name] [hint-expression] \  
  capture-statement`
- `#pragma omp critical [name] [hint-expression] \  
  capture-statement`
- `#pragma omp critical [name] [hint-expression] \  
  write-statement`
- `#pragma omp critical [name] [hint-expression] \  
  update-statement`

**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.

- `#pragma omp barrier`
- `#pragma omp barrier`
- `#pragma omp barrier`

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

- `#pragma omp taskwait [clause[, clause]...]`
- `#pragma omp taskwait [clause[, clause]...]`
- `#pragma omp taskwait [clause[, clause]...]`

**taskgroup** [2.13.7] [2.13.5]  
Specifies a wait on the completion of child tasks of the current task, and waits for descendant tasks.

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

**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.

- `#pragma omp flush [memory-order-clause] [list]`
- `#pragma omp flush [memory-order-clause] [list]`

**ordered** [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.

- `#pragma omp ordered [clause[, clause]...]  
  structured-block`
- `#pragma omp ordered [clause[, clause]...]  
  [memory-order-clause]`
- `#pragma omp ordered [clause[, clause]...]  
  [memory-order-clause]`

**Continued in next column**
Directives and Constructs (continued)

depobj [2.17.10.1]
Stand-alone directive that initializes, updates, or destroys an OpenMP dependency object.

- #pragma omp depobj (depedge clause)
- $omp depobj (depedge 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.

- #pragma omp cancel (constructor-type-clause) [if-clause]
- $omp cancel (constructor-type-clause) [if-clause]


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.

- #pragma omp cancellation point (constructor-type-clause)
- $omp cancellation point (constructor-type-clause)

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.

- #pragma omp threadprivate (list)
- $omp threadprivate (list)

Data environment directive
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 : type-list) (initializer-clause)
- $omp declare reduction & (reduction-identifier : type-list) (initializer-clause)

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 n_threads.

- void omp_set_num_threads (int n_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.

- int omp_get_num_threads (void);

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.

- int omp_in_parallel (void);

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.

- void omp_get_dynamic (void);

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.

- int omp_get_cancellation (void);

omp_set_schedule [3.12.12] [3.12.12]
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 (schedule-kind t_kind, int chunk_size);

omp_get_schedule [3.12.11] [3.12.11]
Returns whether the scheduled parallelism is enabled or disabled, according to the value of the max-active-levels-var ICV.

- void omp_get_schedule (schedule-kind t_kind);

omp_set_nested [3.12.10] [3.12.10]
Enables or disables nested parallelism, by setting the max-active-levels-var ICV.

- void omp_set_nested (int nested);

omp_get_nested [3.12.11] [3.12.11]
Returns whether nested parallelism is enabled or disabled, according to the value of the max-active-levels-var ICV.

- void omp_get_nested (int);

omp_set_schedule [3.12.12] [3.12.12]
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 (schedule-kind t_kind, int chunk_size);

omp_get_schedule [3.12.11] [3.12.11]
Returns whether the scheduled parallelism is enabled or disabled, according to the value of the max-active-levels-var ICV.

- void omp_get_schedule (schedule-kind t_kind);

See omp_get_schedule for kind.

Continued
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_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
int omp_get_active_level (void);
```

**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
int omp_in_final (void);
```

**omp_get_schedule** [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
int (kind=omp_proc_bind_kind) function omp_get_proc_bind (void);
```

**omp_get_level** [3.2.21] [3.2.20]
Returns the number of active levels of parallelism supported.

```c
int omp_get_level (void);
```

**omp_get_schedule** [3.2.23] [3.2.22]
Returns the number of processors available to the execution environment in the specified place.

```c
int omp_get_num_places (int place_num);
```

**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
int omp_get_ancestor_thread_num (int level);
```

**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
int omp_get_num_places (void);
```

**omp_get_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
void omp_set_max_active_levels (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
int omp_get_max_active_levels (void);
```

**omp_get_thread_limit** [3.2.14] [3.2.13]
Returns the value of the thread-limit-var ICV, which is the maximum number of OpenMP threads available.

```c
int omp_get_thread_limit (void);
```

**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
void omp_get_partition_place_nums (int *place_nums);
```

**omp_capture_affinity** [3.2.27] [3.2.26]
Returns the thread affinity policy to be used for the places that do not specify a proc_bind clause.

```c
int (kind=omp_proc_bind_kind) function omp_capture_affinity (void);
```

**omp_get_num_devices** [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
int omp_get_team_size (int level);
```

**omp_get_num_devices** [3.2.30] [3.2.29]
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_partition_num_places** [3.2.27] [3.2.26]
Returns the number of places in the place partition of the innermost implicit task.

```c
int omp_get_partition_num_places (void);
```

**omp_get_num_devices** [3.2.36] [3.2.35]
Returns the number of target devices.

```c
int omp_get_num_devices (void);
```

**omp_get_device_num** [3.2.37]
Returns the device number of the device on which the calling thread is executing.

```c
int omp_get_device_num (void);
```
Runtime Library Routines (continued)

**omp_get_num_teams** [3.3.3] [3.3.32]
Returns the number of teams in the current teams region, or 1 if called from outside of a teams region.

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

**omp_get_team_num** [3.3.3] [3.3.33]
Returns a device number representing the host device.

```c
integer omp_get_team_num (void);
```

**omp_is_initial_device** [3.3.3] [3.3.34]
Returns true if the current task is executing on the host device; otherwise, it returns false.

```c
integer omp_is_initial_device (void);
```

**omp_get_initial_device** [3.3.4] [3.3.35]
Returns a device number representing the host device.

```c
integer omp_get_initial_device (void);
```

**omp_get_max_team_priority** [3.3.4] [3.3.36]
Returns the maximum value that can be specified in the priority clause.

```c
integer omp_get_max_team_priority (void);
```

**omp_pause_resource** [3.3.4] [3.3.43]
Allows the runtime to relinquish resources used by OpenMP on the specified device.

```c
void omp_pause_resource (omp_pause_resource_t kind, int device_num);
```

**omp_pause_resource_all** [3.3.4] [3.3.44]
Allows the runtime to relinquish resources used by OpenMP on the specified device.

```c
void omp_pause_resource_all (omp_pause_resource_t 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]
Initializes an OpenMP lock.

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

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

```c
subroutine omp_init_lock (ivar)
integer (kind=omp_lock_kind) ivar
end subroutine
```

```c
subroutine omp_init_nest_lock (ivar)
integer (kind=omp_nest_lock_kind) ivar
end subroutine
```

**Destroy lock** [3.3.3] [3.3.3]
Ensures that the lock is uninitialised.

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

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

```c
subroutine omp_destroy_lock (ivar)
integer (kind=omp_lock_kind) ivar
end subroutine
```

```c
subroutine omp_destroy_nest_lock (ivar)
integer (kind=omp_nest_lock_kind) ivar
end subroutine
```

**Set lock** [3.3.4] [3.3.4]
Sets an OpenMP lock. The calling task region is suspended until the lock is set.

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

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

```c
subroutine omp_set_lock (ivar)
integer (kind=omp_lock_kind) ivar
end subroutine
```

```c
subroutine omp_set_nest_lock (ivar)
integer (kind=omp_nest_lock_kind) ivar
end subroutine
```

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

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

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

```c
subroutine omp_unset_lock (ivar)
integer (kind=omp_lock_kind) ivar
end subroutine
```

```c
subroutine omp_unset_nest_lock (ivar)
integer (kind=omp_nest_lock_kind) ivar
end subroutine
```

**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);
```

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

```c
logical function omp_test_lock (ivar)
integer (kind=omp_lock_kind) ivar
end function
```

```c
logical function omp_test_nest_lock (ivar)
integer (kind=omp_nest_lock_kind) ivar
end function
```

**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** [3.3.1] [3.3.1]
Returns elapsed wall clock time in seconds.

```c
double omp_get_wtime (void);
```

```c
double precision function omp_get_wtime ()
```

**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** [3.3.1]
Fulfills and destroys an OpenMP event.

```c
void omp_fulfill_event (omp_event_handle_t event);
```

```c
subroutine omp_fulfill_event (event)
integer (kind=omp_event_handle_kind) event
end subroutine
```

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

**omp_target_alloc** [3.4.1] [3.5.1]
Allocates memory in a device data environment.

```c
void *omp_target_alloc (size_t size, int device_num);
```

```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
void *omp_target_memcpy (void *dst, const void *src, size_t length, int dst_device_num, int src_device_num);
```

```c
void *omp_target_memcpy (void *dst, const void *src, size_t length, int dst_device_num, int src_device_num);
```

```c
void *omp_target_memcpy (void *dst, const void *src, size_t length, int *dst_device_num, int src_device_num);
```

**omp_target_mempyc** [3.6.5] [3.5.5]
Copies a rectangular subvolume from a multi-dimensional array to another multi-dimensional array.

```c
void *omp_target_memcpy_rect (void *dst, const void *src, size_t *src_volume, const size_t *src_oferd, int *dst_device_num, int src_device_num);
```

```c
void *omp_target_memcpy_rect (void *dst, const void *src, size_t *src_volume, const size_t *src_oferd, 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
int omp_target_associate_ptr (const void *host_ptr, const void *device_ptr, size_t size, int device_offset, int device_num);
```

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

```c
int omp_target_associate_ptr (const void *host_ptr, const void *device_ptr, size_t size, int device_offset, int device_num);
```
### 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 {omp_c oppose}`
- `integer omp_alloctrait_kind` (For)

### omp_init_allocor [3.7.2]

Initializes allocator and associates it with a memory space.

```c
omp_allocator_handle_t omp_init_allocor ( 
  {omp_mempool_t operator|mespmem, 
  int traits, const omp_alloctrait_t traits(})
```

### omp_destroy_allocor [3.7.3]

Releases all resources used by the allocator handle.

```c
void omp_destroy_allocor ( 
  omp_allocator_handle_t allocator)
```

### omp_alloc [3.7.6]

Requests a memory allocation from a memory allocator.

```c
void *omp_alloc (size_t size, 
  const omp_allocator_t allocator)
```

#### Tool control routine

#### omp_control_tool [3.8]

Enables a program to pass commands to an active tool.

```c
int omp_control_tool (int command, int modifier, 
  void *arg);
```

### 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

## Clauses

### Allocate Clause [2.11.4]

 allocate ([allocator] list)

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

- C/C++: Expression of type `omp_allocator_handle_t`
- For: `integer expression of type in integer`

### 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.
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, mutextinoutset, or inout dependence-type list.
- **mutextinoutset**: 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 mutextinoutset dependence-type on a construct from which a sibling task was previously generated, then the sibling task 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 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-logical-expression
}
```

**Map Clause (2.15.1) [2.15.5.1]**

```
map [[map-type-modifier[.] map-type-modifier[], ... ] map-type : locators]
```

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.

**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-identification: 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: +, -, *, |, \..., &&, \|, ^, \&\& or one of the following intrinsic procedure names: max, min, land, lor, leor,
```

```
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 (i)
```

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
```

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

```
otinbranch
```

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

**Tasking Clauses (2.10) [2.9]**

```
affinity { (off-modifier) [location-list]
```

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

```
allocate { (allocator) : list
```

See Allocate Clause, page 9 of this guide.

```
collapse (i)
```

See SIMD Clauses on this page.

```
default (shared | none) C++
```

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

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

```
depend { (depend-modifier) [dependence-type : location-list
```

See Depend Clause, page 9 of this guide.

```
final { (scalar-expression)
```

```
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)
```

```
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
```

```
identifiers: 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++
```

```
iterator-type: A type specifier. For
```

Notes
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>Implementation-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>Implementation defined.</td>
<td>omp_set_nested()</td>
<td>omp_get_nested()</td>
<td>[6.9] [4.4]</td>
</tr>
<tr>
<td>nthreads-var</td>
<td>OMP_NUM_THREADS</td>
<td>Implementation defined list.</td>
<td>omp_set_num_threads()</td>
<td>omp_get_max_threads()</td>
<td>[6.2] [4.2]</td>
</tr>
<tr>
<td>num-sched-var</td>
<td>OMP_SCHEDULE</td>
<td>Implementation 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>Implementation defined.</td>
<td>(none)</td>
<td>(none)</td>
<td>--</td>
</tr>
<tr>
<td>bind-var</td>
<td>OMP_PROC_BIND</td>
<td>Implementation defined list.</td>
<td>(none)</td>
<td>omp_get_proc_bind()</td>
<td>[6.4] [4.4]</td>
</tr>
<tr>
<td>stacksize-var</td>
<td>OMP_STACKSIZE</td>
<td>Implementation defined.</td>
<td>(none)</td>
<td>(none)</td>
<td>[6.6] [4.7]</td>
</tr>
<tr>
<td>wait-policy-var</td>
<td>OMP_WAIT_POLICY</td>
<td>Implementation 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>Implementation defined.</td>
<td>thread_limit_clause</td>
<td>omp_get_thread_limit()</td>
<td>[6.10] [4.10]</td>
</tr>
<tr>
<td>max-active-levels-var</td>
<td>OMP_MAX.getActive_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>Implementation 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>false</td>
<td>(none)</td>
<td>(none)</td>
<td>[6.13]</td>
</tr>
<tr>
<td>affinity-format-var</td>
<td>OMP_AFFINITY_FORMAT</td>
<td>Implementation defined.</td>
<td>omp_set_affinity_format()</td>
<td>omp_get_affinity_format()</td>
<td>[6.14]</td>
</tr>
<tr>
<td>default-device-var</td>
<td>OMP_DEFAULT_DEVICE</td>
<td>Implementation 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>max-task-priority-var</td>
<td>OMP_MAX_TASK_PRIORITY</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.

<table>
<thead>
<tr>
<th>Environment Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OMP_ALLOCATOR</td>
<td>arg [6.21]</td>
</tr>
<tr>
<td></td>
<td>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):</td>
</tr>
<tr>
<td></td>
<td>omp_default_mem_alloc, omp_low_lat_mem_alloc, omp_large_cap_mem_alloc, omp_const_mem_alloc, omp_high_bw_mem_alloc, omp_thread_mem_alloc</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>alloca_trail</td>
</tr>
<tr>
<td></td>
<td>sync_font</td>
</tr>
<tr>
<td></td>
<td>access</td>
</tr>
<tr>
<td></td>
<td>pool_size</td>
</tr>
<tr>
<td></td>
<td>fallback</td>
</tr>
<tr>
<td></td>
<td>pinned</td>
</tr>
<tr>
<td></td>
<td>partition</td>
</tr>
<tr>
<td></td>
<td>OMP_AFFINITY_FORMAT</td>
</tr>
<tr>
<td></td>
<td>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).</td>
</tr>
<tr>
<td></td>
<td>t</td>
</tr>
<tr>
<td></td>
<td>T</td>
</tr>
<tr>
<td></td>
<td>L</td>
</tr>
<tr>
<td></td>
<td>P</td>
</tr>
<tr>
<td></td>
<td>H</td>
</tr>
<tr>
<td></td>
<td>native_thread_id</td>
</tr>
<tr>
<td></td>
<td>OMP_CANCELLATION</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>OMP_DEBUG</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>OMP_DEFAULT_DEVICE</td>
</tr>
<tr>
<td></td>
<td>Sets the default_device-var ICV that controls the default device number to use in device constructs.</td>
</tr>
<tr>
<td></td>
<td>OMP_DISPLAY_AFFINITY</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>OMP_DISPLAY_ENV</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>OMP_DYNAMIC</td>
</tr>
<tr>
<td></td>
<td>Sets the dyn-var ICV. If TRUE, the implementation may dynamically adjust the number of threads to use for executing parallel regions.</td>
</tr>
</tbody>
</table>
### Environment Variables (continued)

**OMP_MAX_ACTIVE_LEVELS**

Sets the max-active-level-var ICV that controls the maximum number of nested active parallel regions.

**OMP_MAX_TASK_PRIORITY**

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

- **OMP_NESTED**
  Controls nested parallelism with max-active-levels-var ICV.

**OMP_NUM_THREADS**

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

**OMP_PLACES**

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**

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**

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**

Sets the stack-size-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 not specified, size is measured in kilobytes (K).

**OMP_TARGET_OFFLOAD**

Sets the initial value of the target-offload-var ICV. The argument must be one of MANDATORY, DISABLED, or DEFAULT.

**OMP_THREAD_LIMIT**

Sets the thread-limit-var ICV that controls the number of threads participating in the OpenMP program.

**OMP_TOOL**

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_WAIT_POLICY**

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**

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**

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**

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**

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_set_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.”