OpenMP API 5.0

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.

Directive 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**

**Metadirective**

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.

```plaintext
#pragma omp metadirective [clause[ [, clause] ... ]]stmt(s)
```

**Declare variant**

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

```plaintext
#pragma omp declare variant[variant-func-id] clause[ [, clause] ... ]stmt(s)
```

**Requires directive**

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

```plaintext
#pragma omp requires clause[ [, clause] ... ]stmt(s)
```

**Parallel construct**

**Parallel**

**Declares a team of OpenMP threads that execute the region.**

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

**Teams construct**

**Teams**

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

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

**Worksharing constructs**

**Sections**

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

---

© 2019 OpenMP ARB

OMPS15-01-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 \( \text{chunk}_\text{size} \times \text{simd_width} \), 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 [, ]clause] ...
```

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

```
$omp simd [clause [, ]clause] ...
```

**do-loops**

```
$omp simd end do
```

```
$omp simd end simd
```

### 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 for simd [clause [, ]clause] ...
```

**clause:**
- function definition or declaration

```
#pragma omp for simd decl [proc-name] [clause [, ]clause] ...
```

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

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

```
$omp simd [clause [, ]clause] ...
```

```
$omp simd end do
```

```
$omp simd end simd
```

### distribute loop constructs

### distribute [2.8.4] [2.10.8]

Specifies loops which are executed by the initial teams.

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

```
$omp distribute [clause [, ]clause] ...
```

```
$omp end distribute
```

**clause:**
- private (list)
- firstprivate (list)
- lastprivate (list)
- collapse (n)
- dist_schedule (kind, chunk_size)
- allocate (locator-list : list)

### distribute simd [2.9.4.2] [2.10.9]

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

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

```
$omp distribute simd [clause [, ]clause] ...
```

```
$omp end distribute simd
```

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

### Distribute Parallel Worksharing-Loop

### [2.9.4.3] [2.10.10]

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

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

```
$omp simd parallel do [clause [, ]clause] ...
```

```
$omp end simd parallel do
```

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

### Distribute Parallel Worksharing-Loop SIMD

### [2.9.4.4] [2.10.11]

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

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

```
$omp simd parallel do [clause [, ]clause] ...
```

```
$omp end simd parallel do
```

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

### scan directive

### scan [2.9.6]

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

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

```
$omp scan [clause [, ]clause] ...
```

**clause:**
- loop-associated-directive
  - for, for-header
  - structed-block
    - #pragma omp scan clause
      - structured-block
        - end-loop-associated-directive
- end-loop-associated-directive: For do (end do) do simd (end simd) simd (end simd)

### Tasking constructs

### task [2.10.1] [2.9.1]

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

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

```
$omp task [clause [, ]clause] ...
```

```
$omp end task
```

**clause:**
- untied, mergeable
  - private (list), firstprivate (list), shared (list)
  - in_reduction (reduction-identifier : list)
  - depend ([depend-modifier, ] dependence-type : [locator-list])
  - priority (priority-value)
  - allocate (locator-list : list)
    - where off-modifier is iteror (iterators-definition)
  - detach (event-handle)
    - where event-handle is of: type event_handle_t
      - kind omp_event_handle_kind

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

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

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

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

```
#pragma omp taskloop end do
```

**clause:**
- final (scalar-expression)
  - if (task : scalar-expression)
  - default (shared | none)
  - if (task : scalar-logica-express)
  - final (scalar-logica-express)
  - shared (none)

### taskloop [2.10.2] [2.9.2]

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

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

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

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

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

**clause:**
- shared (list), private (list)
  - firstprivate (list), lastprivate (list)
  - reduction ([default, ] reduction-identifier : list)
  - in_reduction (reduction-identifier : list)
  - grainsize (grain-size), num_tasks (num-tasks)
  - collapse (n), priority (priority-value)
  - untied, mergeable, nogroup
  - allocate (locator-list : list)

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

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

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

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

**clause:**
- default (shared | none)
  - final (scalar-expression)
  - shared (none)
  - final (scalar-logica-express)
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.

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

- **for-loops**
- **do-loops**
- **for-end taskloop simd**

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

taskyield [2.10.4] [2.11.2]
Specifies that the current task can be suspended in favor of execution of a different task.

```c
#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.

<table>
<thead>
<tr>
<th>Memory space</th>
<th>Storage selection intent</th>
</tr>
</thead>
<tbody>
<tr>
<td>omp_default_mem_space</td>
<td>System default storage.</td>
</tr>
<tr>
<td>omp_large_cap_mem_space</td>
<td>Storage with large capacity.</td>
</tr>
<tr>
<td>omp_cond_mem_space</td>
<td>Storage optimized for variables with constant values.</td>
</tr>
<tr>
<td>omp_high_bw_mem_space</td>
<td>Storage with high bandwidth.</td>
</tr>
<tr>
<td>omp_low_lat_mem_space</td>
<td>Storage with low latency.</td>
</tr>
</tbody>
</table>

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

```c
#pragma omp allocate [list] [clause]
or
#pragma omp allocate [ ] clause
| [ ]allocate statement |
| alloca | (locator) |
| - where alloca is an expression of: |
| C/C++ | | type(omp_allocator) |
| C/C++ | | For |
| kind omp_allocator_handle_kind |
```

Device directives and construct

target data [2.12.2] [2.10.1.1]
Creates a device environment for te extent of the region.

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

- **structured-block**
- **structured-block**
- **structured-block**
- **structured-block**

| clause: | map ([map-type-modifier] [map-type-modifier] ...) |
| map-type | [locator-list] |
| map-type | [locator-list] |
| map-type | [locator-list] |
| map-type | [locator-list] |

#pragma omp target data [clause [ , ]clause ... ]

| clause: | motion-clause or one of: |
| motion-clause | if (target data): scalar-expression |
| motion-clause | device (integer-expression) |
| motion-clause | for (target data): scalar-logical-expression |
| motion-clause | device (scalar-integer-expression) |

| target update [2.12.6] [2.10.5.3] |
| 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 ... ]
```

| clause: | motion-clause or one of: |
| motion-clause | if (target update): scalar-expression |
| motion-clause | device (integer-expression) |
| motion-clause | for (target update): scalar-logical-expression |
| motion-clause | device (scalar-integer-expression) |

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

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

| clause: | map ([map-type-modifier] [map-type-modifier] ...) |
| map-type | [locator-list] |
| map-type | [locator-list] |
| map-type | [locator-list] |
| map-type | [locator-list] |

#pragma omp target exit data [clause [ , ]clause ... ]

| clause: | motion-clause or one of: |
| motion-clause | if (target exit data): scalar-expression |
| motion-clause | device (integer-expression) |
| motion-clause | for (target exit data): scalar-logical-expression |
| motion-clause | device (scalar-integer-expression) |

declare target [2.12.7] [2.10.6]
A declarative directive that specifies that variables, functions, and subroutines are mapped to a device.

```c
#pragma omp declare target declarations-definition-seq
```

| declarations-definition-seq |
| [ ]declarations-definition-seq |
| [ ]declarations-definition-seq |
| [ ]declarations-definition-seq |
| [ ]declarations-definition-seq |

| #pragma omp declare target [extended-list] |
| - or - |
| #pragma omp declare target [extended-list] |

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

Combined constructs

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

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

| 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 with one or more associated loops and no other statements.

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

| 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.

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

| 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.

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

| clause: | Any of the clauses accepted by the parallel directive, with identical meanings and restrictions. |
Parallel Worksharing-Loop SIMD [2.13.5] [2.11.4]
Shortcut for specifying a parallel construct containing only one work-sharing-loop SIMD construct.

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

| 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.

```c
#pragma omp parallel master [clause [ , clause ] ...]
structured-block
```

| Clause: Any clause used for parallel directive with identical meanings and restrictions. |

master taskloop [2.13.7]
Shortcut for specifying a master construct containing a taskloop construct and no other statements.

```c
#pragma omp master taskloop [clause [ , clause ] ...]
for-loops
```

| 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.

```c
#pragma omp master taskloop simd [clause [ , clause ] ...]
structured-block
```

| Clause: Any clause used for taskloop simd directive with identical meanings and restrictions. |

parallel master taskloop [2.11.9]
Shortcut for specifying a parallel construct containing a master taskloop construct and no other statements.

```c
#pragma omp parallel master taskloop [clause [ , clause ] ...]
for-loops
```

| Clause: Any clause used for parallel or master taskloop directives, except the in_reduction clause, with identical meanings and restrictions. |

parallel master taskloop simd [2.11.10]
Shortcut for specifying a parallel construct containing a master taskloop simd construct and no other statements.

```c
#pragma omp parallel master taskloop simd [clause [ , clause ] ...]
for-loops
```

| Clause: Any clause used for parallel or master taskloop simd directives, except the in_reduction clause, with identical meanings and restrictions. |

teams distribute [2.13.11] [2.11.10]
Shortcut for specifying a teams construct containing a distribute construct and no other statements.

```c
#pragma omp teams distribute [clause [ , clause ] ...]
for-loops
```

| Clause: Any accepted by the teams or distribute directives 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.

```c
#pragma omp target parallel for [clause [ , clause ] ...]
for-loops
```

| Clause: Any accepted by the target or parallel for/do directives, except for copyrin, with identical meanings and restrictions. |

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

```c
#pragma omp target parallel do [clause [ , clause ] ...]
do-loops
```

| Clause: Any accepted by the target or parallel for/do directives, except for copyrin, 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.

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

| Clause: Any accepted by the target or parallel loop directives 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.

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

| 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.

```c
#pragma omp target teams [clause [ , clause ] ...]
structured-block
```

| 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 construct distribute and no other statements.

```c
#pragma omp target teams distribute [clause [ , clause ] ...]
for-loops
```

| Clause: Any accepted by the target or teams distribute directives with identical meanings and restrictions. |
Synchronization constructs

**critical** [2.17.2] [2.12.2]
Restricts execution of the associated structured block to a single thread at a time.

```c
#pragma omp critical ([name] [hint ([hint-expression])] [structured-block])
```

**hint-expression**
An integer constant expression that evaluates to a valid synchronization hint

```c
hint-expression: for
```
A constant expression that evaluates to a scalar value with kind sync_hint_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.

```c
#pragma omp barrier
```

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

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

**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.

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

**atomic** [2.17.7] [2.13.6]
Ensures a specific storage location is accessed atomically. May take one of the following seven forms:

```c
```

**flush** [2.17.8] [2.13.7]
Makes a thread's temporary view of memory consistent with memory, and enforces an order on the memory operations of the variables.

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

**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 dependences in a doacross loop nest.
Directives and Constructs (continued)

cancellation [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 construct-type-clause

Pragma omp cancellation point construct-type-clause

Pragma omp cancellation point construct-type-clause

C/C+

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

Pragma omp threadprivate (list)

Pragma omp threadprivate (list)

C/C+

list: [initializer-clause [ , ] initializer-clause]
list: [initializer-clause [ , ] initializer-clause]
list: [initializer-clause [ , ] initializer-clause]

Runtime Library Routines

Execution environment routines

omp_get_num_procs [3.2.25] [3.2.25]
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);

int omp_get_num_procs (void);

int omp_get_num_procs (void);

Parallel, sections, taskgroup, for
if-clause: if ([cancel] : scalar-expression)
for construct-type-clause: parallel, sections, taskgroup, do
if-clause: if ([cancel] : scalar-logical-expression)

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 nthread-var ICV of
the current task to num_threads.

void omp_set_num_threads (int num_threads);

void omp_set_num_threads (int 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);

int omp_get_num_threads (void);

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);

int omp_get_max_threads (void);

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
thread.

int omp_get_thread_num (void);

int omp_get_thread_num (void);

int omp_get_thread_num (void);

Data environment directives

data [2.19.2] [2.15.2]
Declares a data environment variable.

Pragma omp data (declarator)

Pragma omp data (declarator)

Pragma omp data (declarator)

C/C+

declarator: [type] :: [external] ident [ , ]... [external] ident
declarator: [type] :: [external] ident [ , ]... [external] ident
declarator: [type] :: [external] ident [ , ]... [external] ident

declare mapper [2.19.7.3] [2.16]
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) [clause] [ ]...
Pragma omp declare mapper ([mapper-identifier : ]
type-var) [clause] [ ]...
Pragma omp declare mapper ([mapper-identifier : ]
type-var) [clause] [ ]...

omp_get_schedule [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_schedule (void);

int omp_get_schedule (void);

int omp_get_schedule (void);

omp_set_schedule [3.2.12] [3.2.12]
Affects the schedule that is applied when runtime is used as
schedule kind, by setting the value of the run-sched-var ICV.

void omp_set_schedule (int kind, chunk_size);

void omp_set_schedule (int kind, chunk_size);

void omp_set_schedule (int kind, chunk_size);

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-name-list) [clause] [ ]...
Pragma omp declare reduction ([reduction-identifier : ]
type-name-list) [clause] [ ]...
Pragma omp declare reduction ([reduction-identifier : ]
type-name-list) [clause] [ ]...

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);

int omp_get_num_procs (void);

int omp_get_num_procs (void);

omp_set_schedule [3.2.12] [3.2.12]
Affects the schedule that is applied when runtime is used as
schedule kind, by setting the value of the run-sched-var ICV.

void omp_set_schedule (omp_sched_t kind, int chunk_size);

void omp_set_schedule (omp_sched_t kind, int chunk_size);

void omp_set_schedule (omp_sched_t kind, int chunk_size);

See omp_get_schedule for kind.
Runtime Library Routines (continued)

**omp_get_schedule** [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]:
Returns the value of the active-level-vars ICV for the current device, which is the number of active, nested parallel regions on the device enclosing the task containing the call.

```c
int omp_get_active_level (void);
```

**omp_in_final** [3.2.22]:
Returns true if the routine is executed in a final task region; otherwise, it returns false.

```c
int omp_in_final (void);
```

**omp_get_max_active_levels**
Returns the number of active regions that do not specify a proc_bind clause.

```c
void omp_get_max_active_levels (void);
```

**omp_get_level** [3.2.16]:
Returns the value of the levels-var ICV for the current device, which is the number of nested parallel regions on the device that enclose the task containing the call.

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

**omp_get_num_places** [3.2.24]:
Returns the number of places available to the execution environment in the specified place.

```c
int omp_get_num_places (void);
```

**omp_get_num_procs** [3.2.25]:
Returns the number of processors available to the execution environment in the specified place.

```c
int omp_get_num_procs (int place_num);
```

**omp_get_num_places** [3.2.24]:
Returns the number of places available to the execution environment in the list.

```c
int omp_get_num_places (void);
```

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

```c
void omp_get_place_proc_ids (int place_num, int *ids);
```

**omp_get_place** [3.2.26]:
Returns the number of the place to which the encountering thread is bound.

```c
int omp_get_place (void);
```

**omp_get_partition_num_places** [3.2.28]:
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_get_partition_num_places** [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_num_places (int *place_nums);
```
**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_num_teams** [3.2.41] [3.2.35]
Returns a device number representing the host device.

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

**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(int device_num,
                        omp_pause_resource_t *kind,
                        int device_num);
int omp_pause_resource_all(int device_num,
                            omp_pause_resource_t *kind);
```

**omp_pause_resource** [kind=omp_pause_resource_t]
**omp_pause_resource_all** [kind=omp_pause_resource_t]
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(omp_pause_resource_t *kind, int device_num);
int omp_pause_resource_all(omp_pause_resource_t *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);
```

**Destroy lock** [3.3.2] [3.3.3]
Ensures that the OpenMP lock is uninitialized.

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

**Unset lock** [3.3.3] [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);
```

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

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

**omp_get_wtime** [3.4.2] [3.4.1]
Returns elapsed wall clock time in seconds.

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

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

**omp_fulfill_event** [3.5.1]
Fulfills and destroys an OpenMP event.

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

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

**omp_target_alloc** [3.6.1] [3.5.1]
Allocates memory in a device data environment.

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

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

```c
void *omp_target_memcpy_rect(void *dest,
                             const void *src,
                             size_t element_size,
                             int num dims,
                             const size_t *dest_volume,
                             const size_t *src_volume,
                             const size_t *dest_offsets,
                             const size_t *src_offsets,
                             int dest_device_num,
                             int src_device_num);
```

**omp_targetassociate_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` or have been obtained from implementation-defined runtime routines.

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

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

```c
void *omp_targetdisassociate_ptr(const void *ptr,
                                 int device_num);
```
**Memory management routines**

**Memory Management Types [3.7.1]**

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

```c
enum omp_atv_key_t { integer, character, void* }
enum omp_atv_value_t { integer, real, real64, float, double, void*, pointer };
```


- `omp_atv_key_t` for keys.
- `omp_atv_value_t` for values.

**`omp_init_allocator()`**

Initializes allocator and associates it with a memory space.

```c
void *omp_init_allocator (omp_allocator_handle_t allocator, void *arg);
```

- `allocator`: The memory allocator to be used by allocation calls.
- `arg`: A user-provided pointer.

**`omp_destroy_allocator()`**

Releases all resources used by the allocator handle.

```c
void omp_destroy_allocator (omp_allocator_handle_t allocator);
```

- `allocator`: The memory allocator handle.

**`omp_allocator_handle_t`**

Declares one or more list items to be private to an implicit task to the data environments of the other tasks or loop iterations, establishing dependences only implicitly determined data-sharing attributes to be as specified.

```c
void *omp_alloc (size_t size, omp_allocator_handle_t allocator, void *arg);
```

- `size`: The size of the memory to be allocated.
- `allocator`: The memory allocator handle.
- `arg`: A user-provided pointer.

**`omp_free()`**

Deallocates previously allocated memory.

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

- `ptr`: The pointer to the memory to be deallocated.
- `allocator`: The memory allocator handle.

**`omp_set_default_allocator()`**

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

```c
void *omp_set_default_allocator (omp_allocator_handle_t allocator);
```

- `allocator`: The memory allocator handle.

**`omp_get_default_allocator()`**

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

```c
void *omp_get_default_allocator (void);
```

- `allocator`: The memory allocator handle.

**`omp_control_tool()`**

Enables a program to pass commands to an active tool.

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

- `command`: The command to be executed.
- `modifier`: The modifier to be applied.
- `arg`: A user-provided pointer.

**`omp_control_tool_start()`**

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

```c
void omp_control_tool_start (void);
```

- `command`: The command to be executed.

**`omp_control_tool_end()`**

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

```c
void omp_control_tool_end (void);
```

- `command`: The command to be executed.

**Clauses**

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

**Allocate Clause [2.11.4]**

- `allocate` (locator-list) [list]
  - Specifies the memory allocator to be used to obtain storage for private variables of a directive.
  - `locator`: C/C++ Expression of type `omp_allocator_handle_t`
  - `list`: Integer expression of kind `omp_allocator_handle_t`

**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 other implicit tasks belonging to the parallel region.

**Data Sharing Attribute Clauses [2.19.4] [2.15.3]**

- `shared` (shared | none) C/C++
  - Declares list items to be shared by tasks generated by parallel, teams, or task-generating constructs.
  - `shared`: 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.

- `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
    - C/C++
      - scalar, aggregate, pointer, allocatable

**Depend Clause [2.17.11] [2.13.9]**

- `depend` (dependence-type)
  - `dependence-type` must be source.
  - `dependence-type` must be depend (dependence-type: vector)
    - `dependence-type`: must be sink and vec is the iteration vector with form: x1 [± d1], x2 [± d2], . . . , xn [± dn]
  - Depend modifier: `depend-modifier` (depend-modifier: dependence-type: locator-list)
    - `depend-modifier`: Iterator (locators-definition)
      - dependence-type: in, out, inout, mutexinoutset, depobj
        - in: The generated task will be a dependent task of all previously generated sibling tasks that reference at least one of the list items in an out or inout dependence-type list.
Clauses (continued)

Depend (continued)

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

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

• depobj: The task dependencies 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 constants, it only applies to the semantics of the construct named in the directive-name-modifier if one is specified. If none is specified for a combined or composite construct then the if clause applies to all constructs to which an if clause can apply.

if [[directive-name-modifier ; scalar-expression] C/C++
if [[directive-name-modifier ; scalar-logical-expression] For

Map Clause [2.19.7.1] [2.15.5.1]

map [[map-type-modifier,] [map-type-modifier, ...] ... map-type-modifier] 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 [ ( ) ]
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 with 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 (list)
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 [off-modifier ; ] locator-list
A hint to execute closely to the location of the list items. off-modifier is iterator (iterators-definition). (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) For
See Data Sharing Attribute Clauses, page 9 of this guide.

depend [[depend-modifier ; | ] dependence-type : locator-list]
See Depend Clause, page 9 of this guide. (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 (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 (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.)

if ( [ ] task ; ) scalar-expression C/C+
if ( [ ] task ; ) scalar-logical-expression For
Also see If Clause on this page.

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

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

mergable
Specifies that the generated task is a mergable 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.

reduction [[ 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]

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

iterator (iterators-definition
iterators-definition: iterator-specifier [. iterators-definition ]
iterator-specifier: { iterator-type } identifier = range-specification

identifier: A base language identifier.
range-specification: begin : end [ ; step ]
begin, end: Expressions for which their types can be converted to iterator-type.
step: An integral expression.
iterator-type: A type name. C/C+
viewport: A type specifier. For
Internal Control Variables (ICV) Values

Host and target device ICVs are initialized before OpenMP API constructs or routines execute. After initial values are assigned, the values of environment variables set by the user are read and the associated ICVs for the host device are modified accordingly. The method for initializing a target device’s ICVs is implementation defined.

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

<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>0</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_nest()</td>
<td>omp_get_nest()</td>
<td>[6.9] [4.6]</td>
</tr>
<tr>
<td>nthreads-var</td>
<td>OMP_NUM_THREADS</td>
<td>Implemented list.</td>
<td>omp_set_nthreads()</td>
<td>omp_get_max_threads()</td>
<td>[6.2] [4.2]</td>
</tr>
<tr>
<td>run-sched-var</td>
<td>OMP_SCHEDULER</td>
<td>Implemented defined.</td>
<td>omp_get_schedule()</td>
<td>[6.1] [4.1]</td>
<td></td>
</tr>
<tr>
<td>def-sched-var</td>
<td>(none)</td>
<td>Implemented def.</td>
<td>(none)</td>
<td>(none)</td>
<td>---</td>
</tr>
<tr>
<td>bind-var</td>
<td>OMP_PROC_BIND</td>
<td>Implemented 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>[6.4] [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>(none)</td>
<td>thread_limit</td>
<td>[6.10] [4.10]</td>
</tr>
<tr>
<td>max-active-levels-var</td>
<td>OMP_MAX_ACTIVE_LEVELS,</td>
<td>Implementation defined.</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>(none)</td>
<td>---</td>
</tr>
<tr>
<td>levels-var</td>
<td>(none)</td>
<td>zero</td>
<td>(none)</td>
<td>(none)</td>
<td>---</td>
</tr>
<tr>
<td>place-partition-var</td>
<td>OMP_PLACES</td>
<td>Implementation defined.</td>
<td>(none)</td>
<td>(none)</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>Implementation defined.</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_device()</td>
<td>omp_get_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>Implementation defined.</td>
<td>(none)</td>
<td>omp_set_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>(none)</td>
<td>omp_set_default_allocator()</td>
<td>omp_get_default_allocator()</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 below (for details, see table 2.10):

- omp_default_mem_alloc
- omp_large_cap_mem_alloc
- omp_const_mem_alloc
- omp_high_bw_mem_alloc

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

**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 %[([0-9] size ) type, where the field type may be either the short or long names listed below (Table 6.2).

- t team_num
- T num_teams
- L nesting_level
- P process_id
- H host

**OMP_CANCELLATION var [6.11]**

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

**OMP_DEBUG var [6.20]**

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

**OMP_DEFAULT_DEVICE device [6.15]**

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]**

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 variable=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]**

Sets the dyn-var ICV. var may be TRUE or FALSE. If TRUE, the implementation may dynamically adjust the number of threads to use for executing parallel regions.
Environment Variables (continued)

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OMP_MAX_ACTIVE_LEVELS</td>
<td>Sets the maximum number of active parallel regions.</td>
</tr>
<tr>
<td>OMP_TASK_PRIORITY</td>
<td>Sets the task-priority ICV that controls the use of task priorities.</td>
</tr>
<tr>
<td>OMP NESTED</td>
<td>Controls nested parallelism with max-active-levels ICV.</td>
</tr>
<tr>
<td>OMP_NUM_THREADS</td>
<td>Sets the number of threads to use for parallel regions.</td>
</tr>
<tr>
<td>OMP PLACES</td>
<td>Sets the place-partition ICV that defines the OpenMP parallelism.</td>
</tr>
<tr>
<td>OMP_PROC_BIND</td>
<td>Sets the thread affinity policy to use for parallel regions at the</td>
</tr>
<tr>
<td></td>
<td>corresponding nested level. policy can be the values true, false, or</td>
</tr>
<tr>
<td></td>
<td>comma-separated list of master, close, or spread in quotes.</td>
</tr>
<tr>
<td>OMP_SCHEDULE</td>
<td>Activates an OMPT tool.</td>
</tr>
<tr>
<td></td>
<td>Provides the name of a dynamically linked library appropriate for the</td>
</tr>
<tr>
<td></td>
<td>architecture and operating system used by the application in the</td>
</tr>
<tr>
<td></td>
<td>tool-libraries-var ICV.</td>
</tr>
<tr>
<td>OMP_STACKSIZE</td>
<td>Sets the stacksize-var ICV that specifies the size of the stack for</td>
</tr>
<tr>
<td></td>
<td>threads created by the OpenMP implementation.</td>
</tr>
<tr>
<td>OMP_TARGET_OFFLOAD</td>
<td>Sets the initial value of the target-offload-var ICV.</td>
</tr>
<tr>
<td></td>
<td>The arg must be one of MANDATORY, DISABLED, or DEFAULT.</td>
</tr>
</tbody>
</table>

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_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.

Learn More About OpenMP

OpenMPCon

Developer’s Conference

Held back-to-back with IWOMP, the annual OpenMP 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, 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 © 2019 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.”