Directives and Constructs

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

variant directives

Metadata directives [2.3.4]

A directive that can specify multiple directive variants of which one may be conditionally selected to replace the metadirective based on the enclosing OpenMP context.

```c
#pragma omp metadirective [clause[ [, ]clause] ... ]
- or -
#pragma omp begin metadirective [clause[ [, ]clause] ... ]
#pragma omp end metadirective
```

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

declare variant [2.3.5]

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

```c
#pragma omp declare variant[variant-func-id] clause
#pragma omp declare variant[variant-func-id] clause
```

clause: function definition or declaration

```c
#pragma omp declare variant [ &
 [base-proc-name]: [variant-proc-name] ]
```

clause: match ([context-selector-specification])

variant-func-id: C/C++

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

variant-proc-name: For

The name of a function variant that is a base language identifier.

requires directive

Requires directives [2.4]

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

```c
#pragma omp requires [  ] clause
```

clause: reverse_offload

unified_address

unified_shared_memory

atomic_default_mem_order([seq_cst | acq_rel | relaxed])

dynamic_allocators

parallel construct

parallel [2.6] [2.5]

Creates a team of OpenMP threads that execute the region.

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

clause:

```c
private (list), firstprivate (list), lastprivate ([lastprivate-modifier]: list)
```

```c
reduction ([reduction-modifier], [reduction-identifier]: list)
```

```c
allocate ([allocator]: list)
```

```c
C/C++ nowait
```

single [2.8.2] [2.7.3]

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

```c
#pragma omp single [clause[ [, ]clause] ... ]
```

clause:

```c
private (list), firstprivate (list), lastprivate ([lastprivate-modifier]: list)
```

```c
allocate ([allocator]: list)
```

```c
C/C++ nowait
```

teams construct

teams [2.7] [2.10.7]

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

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

```c
private (list), firstprivate (list), shared (list)
```

```c
reduction ([default]: [reduction-identifier]: list)
```

```c
allocate ([allocator]: list)
```

```c
C/C++ nowait
```

workshare [2.8.3] [2.7.4]

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

```c
#pragma omp workshare structured-block
```

```c
C/C++ nowait
```

Worksharing-loop construct

for / do [2.9.2] [2.7.1]

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

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

```c
for-loops
```

```c
do-loops
```

```c
C/C++ nowait
```

Worksharing constructs

sections [2.8.1] [2.7.2]

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

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

```c
C/C++
```

```c
nowait
```

kind:

- static: Iterations are divided into chunks of size chunk_size and assigned to threads in the team in round-robin fashion in order of thread number.
- dynamic: Each thread executes a chunk of iterations then requests another chunk until none remain.
- guided: Each thread executes a chunk of iterations then requests another chunk until no chunks remain to be assigned. Chunk size is different for each chunk, with each successive chunk smaller than the last.
## Directives and Constructs (continued)

### SIMD directives

**simd** [2.9.3.1] [2.8.1]

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

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

For loops:

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

Safepen:

```c
safepen (length), simdlen (length)
```

Aligned:

```c
aligned (list : alignment)
```

Non-temporal:

```c
safelen (length)
```

```

Clause:

```c
private (list) , firstprivate (list)
```

```

Collapse:

```c
collapse (n)
```

```

List:

```c
list
```

```

Unspecified:

```c
in_reduction (reduction-identifier)
```

```

```

For:

```c
for (scalar-logical-expression)
```

```
```

### Worksharing-Loop SIMD [2.9.3.2] [2.8.3]

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

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

Do loops:

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

Clause:

```c
safepen (length), simdlen (length)
```

```
```

### 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 of a SIMD loop.

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

For loops:

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

### Distibute loop constructs

**distribute** [2.9.4.1] [2.10.8]

Specifies loops which are executed by the thread teams.

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

Do loops:

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

Clause:

```c
private (list) , firstprivate (list)
```

```
```

### Distibute simd [2.9.4.2] [2.10.9]

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

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

Do loops:

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

Clause:

```c
private (list) , firstprivate (list)
```

```
```

### Distibute Parallel Worksharing-Loop

[2.9.4.3] [2.10.10]

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

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

Do loops:

```c
$omp distribute parallel do [clause[, clause]...] 
```

Clause:

```c
private (list) , firstprivate (list)
```

```
```

### Distibute Parallel Worksharing-Loop SIMD

[2.9.4.4] [2.10.11]

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

```c
#pragma omp distribute parallel for simd \ 
(clause[, clause]) ... 
```

Do loops:

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

Clause:

```c
private (list) , firstprivate (list)
```

```
```

### Scan directive

**scan** [2.9.6]

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

```c
#pragma omp scan [clause] 
```

Loop-Associated Directives:

```c
loop-associated-directive for-loop-directives
```

```
```

### Tasking constructs

**task** [2.10.1] [2.9.1]

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

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

Structure Block:

```c
structured-block
```

```
```

### Taskloop

**taskloop** [2.10.2] [2.9.2]

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

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

For loops:

```c
$omp taskloop [clause...] 
```

Clause:

```c
shared (list) , private (list)
```

```
```

© 2018 OpenMP ARB
**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 be executed in parallel using OpenMP tasks.

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

**for-loops**

```
1$omp taskloop [clause[ [, clause] ...] ]
```

**do-loops**

```
1$omp end taskloop
```

clause: Any accepted by the task 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.

```
#pragma omp taskyield
```

```
1$omp taskyield
```

### Memory management directive

#### Memory spaces [2.11.1]

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

<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_const_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.

```
#pragma omp allocate [list] [clause]
```

```
1$omp allocate [list] [clause]
```

```
1$omp allocate [list] [clause] [list]
```

```
1$omp allocate [list] clause [list]
```

```
1$omp allocate [list] clause [list]
```

clause:

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

### Device directives and construct

#### target data [2.12.2] [2.10.1]

Creates a device environment for extent of the region.

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

```
1$omp target data [clause[ [, clause] ...]]
```

```
1$omp target data [claue] [clause]
```

```
1$omp target data [clause] [clause]
```

clause:

- **map (locator-list)**
  - use_device_ptr(list), use_device_addr(list)
  - stmt

- **device (locator-list)**

- **foreach (locator-list)**

- **simd (locator-list)**

#### target enter data [2.12.3] [2.10.2]

Maps variables to a device data environment.

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

```
1$omp target enter data [clause[ [, clause] ...]]
```

```
1$omp target enter data [clause] [clause]
```

```
1$omp target enter data [clause] [clause]
```

clause:

- **map (locator-list)**
  - use_device_ptr(list), use_device_addr(list)

- **device (locator-list)**

#### target exit data [2.12.4] [2.10.3]

Unmaps variables from a device data environment.

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

```
1$omp target exit data [clause[ [, clause] ...]]
```

```
1$omp target exit data [clause] [clause]
```

```
1$omp target exit data [clause] [clause]
```

clause:

- **map (locator-list)**
  - use_device_ptr(list), use_device_addr(list)

- **device (locator-list)**

#### target enter construct [2.12.5] [2.10.4]

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

```
#pragma omp target enter construct [clause[ [, clause] ...]]
```

```
1$omp target enter construct [clause[ [, clause] ...]]
```

```
1$omp target enter construct [clause] [clause]
```

```
1$omp target enter construct [clause] [clause]
```

clause:

- **map (locator-list)**
  - use_device_ptr(list), use_device_addr(list)

- **device (locator-list)**

#### target update [2.12.6] [2.10.5]

Specifies the corresponding list items in the device data environment consistent with their original list items, according to the specified motion clauses.

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

```
1$omp target update [clause[ [, clause] ...]]
```

```
1$omp target update [clause] [clause]
```

```
1$omp target update [clause] [clause]
```

clause:

- **motion-clause** or one of:
  - **nowait**: depend 
  - **nowait**: depend 
  - **nowait**: depend 

### Memory spaces [2.11.1]

#### 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.

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

```
1$omp parallel for [clause[ [, clause] ...]]
```

```
1$omp parallel do [clause[ [, clause] ...]]
```

```
1$omp end parallel do
```

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

#### parallel loop [2.13.2]

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

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

```
1$omp parallel loop [clause[ [, clause] ...]]
```

```
1$omp parallel loop [clause[ [, clause] ...]]
```

```
1$omp end parallel loop
```

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

#### parallel sections [2.13.3] [2.11.2]

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

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

```
1$omp parallel sections [clause[ [, clause] ...]]
```

```
1$omp parallel sections [clause[ [, clause] ] ...]
```

```
1$omp parallel sections [clause[ [, clause] ] ...]
```

```
1$omp end parallel sections
```

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

#### parallel workshare [2.13.4] [2.11.3]

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

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

```
1$omp parallel workshare [clause[ [, clause] ...]]
```

```
1$omp parallel workshare [clause[ [, clause] ...]]
```

```
1$omp end parallel workshare
```

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

<table>
<thead>
<tr>
<th><strong>Parallel Worksharing-Loop SIMD</strong> [2.13.5] [2.11.4]</th>
<th>Shortcut for specifying a parallel construct containing one do simd construct and no other statements.</th>
</tr>
</thead>
</table>
| #pragma omp parallel simd do simd \
| [clause[ , ]clause] ... ] for-loops |
| !$omp parallel do simd [clause[ , ]clause] ... ] do-loops |
| !$omp end parallel do simd/ |
| clause: Any accepted by the parallel or for/do simd directives with identical meanings and restrictions. |

<table>
<thead>
<tr>
<th><strong>parallel master</strong> [2.11.6]</th>
<th>Shortcut for specifying a parallel construct containing one master construct and no other statements.</th>
</tr>
</thead>
<tbody>
<tr>
<td>#pragma omp parallel master [clause[ , ]clause] ... ] structured-block</td>
<td></td>
</tr>
<tr>
<td>!$omp parallel master [clause[ , ]clause] ... ] do-loops</td>
<td></td>
</tr>
<tr>
<td>!$omp parallel master taskloop</td>
<td></td>
</tr>
<tr>
<td>$omp master parallel taskloop</td>
<td></td>
</tr>
<tr>
<td>clause: Any accepted by the parallel or taskloop directives with identical meanings and restrictions.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>master taskloop</strong> [2.11.7]</th>
<th>Shortcut for specifying a master construct containing a taskloop construct and no other statements.</th>
</tr>
</thead>
<tbody>
<tr>
<td>#pragma omp parallel master taskloop [clause[ , ]clause] ... ] for-loops</td>
<td></td>
</tr>
<tr>
<td>!$omp parallel master taskloop [clause[ , ]clause] ... ] do-loops</td>
<td></td>
</tr>
<tr>
<td>!$omp end parallel master taskloop</td>
<td></td>
</tr>
<tr>
<td>clause: Any clause used for parallel or master directives with identical meanings and restrictions.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>parallel taskloop</strong> [2.11.8]</th>
<th>Shortcut for specifying a parallel taskloop construct and no other statements.</th>
</tr>
</thead>
<tbody>
<tr>
<td>#pragma omp parallel taskloop [clause[ , ]clause] ... ] for-loops</td>
<td></td>
</tr>
<tr>
<td>!$omp parallel taskloop [clause[ , ]clause] ... ] do-loops</td>
<td></td>
</tr>
<tr>
<td>!$omp end parallel taskloop</td>
<td></td>
</tr>
<tr>
<td>clause: Any clause used for parallel or master taskloop constructs with identical meanings and restrictions.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>target simd</strong> [2.11.9]</th>
<th>Shortcut for specifying a target construct containing a simd construct and no other statements.</th>
</tr>
</thead>
<tbody>
<tr>
<td>#pragma omp target simd [clause[ , ]clause] ... ] for-loops</td>
<td></td>
</tr>
<tr>
<td>!$omp target simd [clause[ , ]clause] ... ] do-loops</td>
<td></td>
</tr>
<tr>
<td>!$omp end target simd</td>
<td></td>
</tr>
<tr>
<td>clause: Any accepted by the target simd directives with identical meanings and restrictions.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>target teams</strong> [2.11.10]</th>
<th>Shortcut for specifying a target construct containing a teams construct and no other statements.</th>
</tr>
</thead>
<tbody>
<tr>
<td>#pragma omp target teams [clause[ , ]clause] ... ] structured-block</td>
<td></td>
</tr>
<tr>
<td>!$omp target teams [clause[ , ]clause] ... ] do-loops</td>
<td></td>
</tr>
<tr>
<td>!$omp end target teams</td>
<td></td>
</tr>
<tr>
<td>clause: Any clause used for target or teams directives with identical meanings and restrictions.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>teams distribute</strong> [2.11.11]</th>
<th>Shortcut for specifying a teams construct containing a construct and no other statements.</th>
</tr>
</thead>
<tbody>
<tr>
<td>#pragma omp teams distribute [clause[ , ]clause] ... ]</td>
<td></td>
</tr>
<tr>
<td>!$omp teams distribute [clause[ , ]clause] ... ] do-loops</td>
<td></td>
</tr>
<tr>
<td>!$omp end teams distribute</td>
<td></td>
</tr>
<tr>
<td>clause: Any accepted by the teams or distribute simd directives with identical meanings and restrictions.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>teams loop</strong> [2.11.12]</th>
<th>Shortcut for specifying a teams construct containing a loop construct and no other statements.</th>
</tr>
</thead>
<tbody>
<tr>
<td>#pragma omp teams loop [clause[ , ]clause] ... ] for-loops</td>
<td></td>
</tr>
<tr>
<td>!$omp teams loop [clause[ , ]clause] ... ] do-loops</td>
<td></td>
</tr>
<tr>
<td>!$omp end teams loop</td>
<td></td>
</tr>
<tr>
<td>clause: Any clause used for teams or distribute parallel for do simd directives with identical meanings and restrictions.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Target Parallel Worksharing-Loop SIMD</strong> [2.11.13] [2.11.5]</th>
<th>Shortcut for specifying a target construct with a parallel worksharing-loop construct and no other statements.</th>
</tr>
</thead>
<tbody>
<tr>
<td>#pragma omp target parallel loop [clause[ , ]clause] ... ] for-loops</td>
<td></td>
</tr>
<tr>
<td>!$omp target parallel loop [clause[ , ]clause] ... ] do-loops</td>
<td></td>
</tr>
<tr>
<td>!$omp end target parallel loop</td>
<td></td>
</tr>
<tr>
<td>clause: Any accepted by the target parallel loop simd directives with identical meanings and restrictions.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>teams distribute</strong> [2.11.14]</th>
<th>Shortcut for specifying a teams construct containing a parallel worksharing-loop simd construct and no other statements.</th>
</tr>
</thead>
<tbody>
<tr>
<td>#pragma omp teams distribute simd [clause[ , ]clause] ... ] for-loops</td>
<td></td>
</tr>
<tr>
<td>!$omp teams distribute simd [clause[ , ]clause] ... ] do-loops</td>
<td></td>
</tr>
<tr>
<td>!$omp end teams distribute simd</td>
<td></td>
</tr>
<tr>
<td>clause: Any accepted by the teams or distribute simd directives with identical meanings and restrictions.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>teams loop</strong> [2.11.15]</th>
<th>Shortcut for specifying a teams construct containing a loop simd construct and no other statements.</th>
</tr>
</thead>
<tbody>
<tr>
<td>#pragma omp teams loop [clause[ , ]clause] ... ] for-loops</td>
<td></td>
</tr>
<tr>
<td>!$omp teams loop [clause[ , ]clause] ... ] do-loops</td>
<td></td>
</tr>
<tr>
<td>!$omp end teams loop</td>
<td></td>
</tr>
<tr>
<td>clause: Any accepted by the teams or loop directives with identical meanings and restrictions.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>target simd</strong> [2.11.16]</th>
<th>Shortcut for specifying a target construct containing a simd construct and no other statements.</th>
</tr>
</thead>
<tbody>
<tr>
<td>#pragma omp target simd [clause[ , ]clause] ... ] for-loops</td>
<td></td>
</tr>
<tr>
<td>!$omp target simd [clause[ , ]clause] ... ] do-loops</td>
<td></td>
</tr>
<tr>
<td>!$omp end target simd</td>
<td></td>
</tr>
<tr>
<td>clause: Any accepted by the target simd directives with identical meanings and restrictions.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>target teams</strong> [2.11.17]</th>
<th>Shortcut for specifying a target construct containing a teams construct and no other statements.</th>
</tr>
</thead>
<tbody>
<tr>
<td>#pragma omp target teams [clause[ , ]clause] ... ] structured-block</td>
<td></td>
</tr>
<tr>
<td>!$omp target teams [clause[ , ]clause] ... ] do-loops</td>
<td></td>
</tr>
<tr>
<td>!$omp end target teams</td>
<td></td>
</tr>
<tr>
<td>clause: Any accepted by the target teams directives with identical meanings and restrictions.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>teams distribute</strong> [2.11.18]</th>
<th>Shortcut for specifying a teams construct containing a teams construct and no other statements.</th>
</tr>
</thead>
<tbody>
<tr>
<td>#pragma omp target teams distribute [clause[ , ]clause] ... ]</td>
<td></td>
</tr>
<tr>
<td>!$omp target teams distribute [clause[ , ]clause] ... ] do-loops</td>
<td></td>
</tr>
<tr>
<td>!$omp end target teams distribute</td>
<td></td>
</tr>
<tr>
<td>clause: Any accepted by the target teams distribute directives with identical meanings and restrictions.</td>
<td></td>
</tr>
</tbody>
</table>
Synchronization constructs

**critical** [2.17.1] [2.13.2]
Restricts execution of the associated structured block to a single thread at a time.

```
#pragma omp critical ([name] [, hint [hint-expression]])
```

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

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

**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
```

**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] ...]
```

```
!$omp taskwait [clause [, clause] ...]
```

**taskgroup** [2.17.6] [2.13.5]
Specifies a wait on the completion of child tasks of the current task, and waits for descendant tasks.

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

```
!$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:

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

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

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

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

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

**ordered** [2.17.9] [2.13.8]
Specifies a structured block in a worksharing-loop, simd, or worksharing-loop SIMD region, or it specifies cross-iteration dependences in a doacross loop nest.
Directives and Constructs (continued)

**depmobj** [2.17.10.1]**

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

```c
#pragma omp depmobj (depmobj) clause
```

**omp_get_nested** [2.19.2] [2.14.2]**

Introduces a user-defined cancellation point at which tasks check if cancellation of the innermost enclosing region of the type specified has been activated.

```c
#pragma omp cancellation point construct-type-clause
```

### Cancellation constructs **cancel** [2.18.1] [2.14.1]**

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

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

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

**Runtime Library Routines**

**omp_get_num_procs** [3.2.5] [3.2.5]**

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

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

**omp_in_parallel** [3.2.6] [3.2.6]**

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

```c
int omp_in_parallel (void);
```

**omp_set_dynamic** [3.2.7] [3.2.7]**

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

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

**omp_get_dynamic** [3.2.8] [3.2.8]**

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

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

**omp_get_cancellation** [3.2.9] [3.2.9]**

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

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

**omp_set_nested** [3.2.10] [3.2.10]**

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

```c
void omp_set_nested (int nested);
```

**omp_get_nested** [3.2.11] [3.2.11]**

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

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

**omp_get_schedule** [3.2.12] [3.2.12]**

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

```c
void omp_get_schedule (omp_sched_t kind, int chunk_size);
```
**Runtime Library Routines (continued)**

- **omp_get_schedule** [3.2.12] [3.2.13]
  - Returns the value of run-sched-var ICV, which is the schedule applied when runtime schedule is used.
  - **void omp_get_schedule (**
    - `omp_sched_t *kind`, int *chunk_size);**

- **omp_get_partition_num_places ()**
  - Returns the value of the partition-num-places ICV, which determines the number of active, nested parallel regions enclosing the task that contains the call.
  - **int omp_get_partition_num_places (void);**

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

- **omp_get_ancestor_thread_num ()**
  - Returns the value of the ancestor thread num ICV, which determines the maximum number of nested active parallel regions.
  - **int omp_get_ancestor_thread_num (void);**

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

- **omp_get_partition_place_nums ()**
  - Returns the list of place numbers corresponding to the places in the place-partition-var ICV of the innermost implicit task.
  - **void omp_get_partition_place_nums (**
    - int *place_nums);**

- **omp_get_partition_num_places ()**
  - Returns the value of the partition-num-places ICV, which determines the number of active, nested parallel regions enclosing the task that contains the call.
  - **int omp_get_partition_num_places (void);**

- **omp_get_num_places ()**
  - Returns the value of the place-num-vars ICV, which determines the number of active, nested parallel regions enclosing the task that contains the call.
  - **int omp_get_num_places (void);**

- **omp_get_num_devices ()**
  - For the enclosing device region, returns the number of processors available to the subsequent nested parallel regions that do not specify a proc_bind clause.
  - **int omp_get_num_devices (void);**

- **omp_get_num_active_levels ()**
  - Returns the value of the num_active-level-vars ICV, which determines the number of active, nested parallel regions enclosing the task that contains the call.
  - **int omp_get_num_active_levels (void);**

- **omp_get_num_places ()**
  - Returns the value of the place-num-vars ICV, which determines the number of active, nested parallel regions enclosing the task that contains the call.
  - **int omp_get_num_places (void);**

- **omp_get_max_active_levels ()**
  - Returns the number of processors available to the execution environment in the specified place.
  - **int omp_get_max_active_levels (void);**

- **omp_get_level ()**
  - Returns the value of the level-vars ICV, which is the number of active, nested parallel regions enclosing the task containing the call.
  - **int omp_get_level (void);**

- **omp_get_num_devices ()**
  - Returns the value of the device_num-vars ICV, which determines the number of active, nested parallel regions enclosing the task that contains the call.
  - **int omp_get_num_devices (void);**

- **omp_get_max_active_levels ()**
  - Returns the value of the max_active-level-vars ICV, which determines the number of active, nested parallel regions enclosing the task containing the call.
  - **int omp_get_max_active_levels (void);**

- **omp_get_ancestor_thread_num ()**
  - Returns the value of the ancestor thread num ICV, which determines the number of active, nested parallel regions enclosing the task containing the call.
  - **int omp_get_ancestor_thread_num (void);**

- **omp_get_device_num ()**
  - Returns the value of the device_num-vars ICV, which determines the number of active, nested parallel regions enclosing the task that contains the call.
  - **int omp_get_device_num (void);**

- **omp_get_team_size ()**
  - Returns the value of the team-size-vars ICV, which determines the number of active, nested parallel regions enclosing the task that contains the call.
  - **int omp_get_team_size (void);**

- **omp_get_place_num_procs ()**
  - Returns the value of the place_num_procs-vars ICV, which determines the number of active, nested parallel regions enclosing the task that contains the call.
  - **int omp_get_place_num_procs (void);**

- **omp_get_place_proc_ids ()**
  - Returns the value of the place_proc_ids-vars ICV, which determines the number of active, nested parallel regions enclosing the task that contains the call.
  - **int omp_get_place_proc_ids (void);**

- **omp_get_num_devices ()**
  - Returns the value of the num_devices-vars ICV, which determines the number of active, nested parallel regions enclosing the task that contains the call.
  - **int omp_get_num_devices (void);**
Runtime Library Routines (continued)

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

```
void omp_init_lock_with_hint (omp_lock_t *lock, omp_sync_hint_t hint);
void omp_init_nest_lock_with_hint (omp_nest_lock_t *lock, omp_sync_hint_t hint);
```

Subroutine omp_init_lock_with_hint (svar; hint)
integer (kind=omp_lock_kind) svar
integer (kind=omp_sync_hint_kind) hint

Subroutine omp_init_nest_lock_with_hint (nvar; hint)
integer (kind=omp_nest_lock_kind) nvar
integer (kind=omp_sync_hint_kind) hint

Hint: see [2.17.12]

Destroy lock [3.3.3] [3.3.3]
Ensure that the OpenMP lock is uninitialized.

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

Subroutine omp_destroy_lock (lock)
integer (kind=omp_lock_kind) lock

Subroutine omp_destroy_nest_lock (nlock)
integer (kind=omp_nest_lock_kind) nlock

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

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

Subroutine omp_unset_lock (lock)
integer (kind=omp_lock_kind) lock

Subroutine omp_unset_nest_lock (nlock)
integer (kind=omp_nest_lock_kind) nlock

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

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

Logical function omp_test_lock (lock)
integer (kind=omp_lock_kind) lock

Logical function omp_test_nest_lock (nlock)
integer (kind=omp_nest_lock_kind) nlock

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 ()
```

Returns the precision of the timer (seconds between ticks) used by omp_get_wtime.

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

```
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 (event_handler)
```

Fulfills and destroys an OpenMP event.

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

Subroutine omp_fulfill_event (event)
integer (kind=omp_event_handle_kind) event

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

```
omp_target_alloc (size_t size, int device_num);
omp_target_free (device_ptr, device_num);
```

Allocates memory in a device data environment.

Copies memory between any combination of host and device pointers.

```
omp_target_memcpy (host_ptr, device_ptr, size_t size, int device_num, int src_device_num, int src_offset, int dst_device_num, int dst_offset, size_t volume, size_t src_offset, size_t src_device_num,
```

```
```

Copies a rectangular subvolume from a multi-dimensional array to another multi-dimensional array.

```
void omp_target_memcpy_rect(void *dst, const void *src, size_t src_length, size_t src_offset, size_t src_device_num, int dst_device_num, int dst_offset, size_t volume, size_t dst_device_num, int dst_offset, size_t dst_data_size, int dst_data_dimensions, size_t src_device_num, size_t src_data_size, int src_data_dimensions);
```

Maps a device pointer, which may be returned from omp_target_alloc or implementation-defined runtime routines, to a host pointer.

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

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

```
void omp_target_disassociate_ptr (const void *ptr, 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]
Initialize an OpenMP lock.

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

Subroutine omp_init_lock (lock)
integer (kind=omp_lock_kind) lock

Subroutine omp_init_nest_lock (nlock)
integer (kind=omp_nest_lock_kind) nlock

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

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

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

Subroutine omp_init_lock (lock)
integer (kind=omp_lock_kind) lock

Subroutine omp_init_nest_lock (nlock)
integer (kind=omp_nest_lock_kind) nlock
Runtime Library Routines (continued)

Memory management routines

Memory Management Types [3.7.2]
The `omp_...`_t struct in C/C++ and omp_...trait type in Fortran define members named key and value, with these types and values:

```c
enum omp_...key_t { ... }  // For
int arg, const omp_...t traits;  // For
omp_atv_X where X may be one of sync, hint, alignment, access, pool_size, fallback, fb_data, pinned, partition
enum omp_...value_t { ... }  // For
```  

```c
region.
function omp_...tool (&arg, ...)
linear-list[...]
dependence-type : vec
region.
:: void
```  

 Explicitly determines default data-sharing attributes to be as specified.

```c
omp_allocator_handle_t omp_...allocate (allocator);
```  

 Deallocates previously allocated memory.

```c
void omp_alloc (size_t size, 
omp_allocator_handle_t _1_);  // For
```  

 Enables a program to pass commands to an active tool.

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

 Tool control routine

Allocating variables

Allocate Clause [2.11.4]

allocate (allocation) list

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

```c
allocate (allocation) list
```  

 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.

```c
copyin (list)
```  

 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.

```c
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.

```c
shared (list)
```  

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

 ```c
private (list)
```  

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

 ```c
firstprivate (list)
```  

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

 ```c
lastprivate (list)
```  

 Declares one or more list items to be private to an implicit task or SIMD lane, and causes the corresponding original list item to be updated after the end of the region.

 ```c
lastprivate-modifier: conditional
```  

 Linear (linear-list: linear-step)

 Declares one or more list items to be private and to have a linear relationship with respect to the iteration space of a loop associated with the construct on which the clause appears.

 ```c
linear-list: list or modifier(list)
```  

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

 Defaultmap Clause [2.19.7.2] [2.15.5.2]

`defaultmap (implicit-behavior: variable-category)`

Explicitly determines the data-mapping attributes referenced in a target construct and would otherwise be implicitly determined.

 ```c
map-type: alloc, to, from, tofrom, firstprivate, none, default
variable-category: C/C++
```  

 scalar, aggregate, pointer

variable-category: scalar, aggregate, pointer, allocatable

 Depend Clause [2.17.11] [2.13.9]

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

 ```c
depend (dependence-type)
```  

 dependence-type may be source.

 ```c
depend (dependence-type : vec)
```  

 dependence-type: sink and is the iteration vector with the form: \( x_1 \times [d_1], x_2 \times [d_2], \ldots, x_n \times [d_n] \)

 ```c
depend ([depend-modifier,dependence-type : locators-list])
```  

 ```c
dependence-modifier: iterator (iterators-definition)
```  

 dependence-type: in, out, inout, mutexinoutset, depobj

 • in: The generated task will be dependent of all previously generated sibling tasks that reference at least one of the list items in an out or inout dependence-type list.
### 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`, `mutxinoutset`, or `inout` dependence-type list.
- **mutxinoutset**: 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 `mutxinoutset dependence-type` on a construct from which a sibling task was previously generated, then the sibling tasks will be mutually exclusive tasks.

### Depend (continued)

- **depen** The task dependences are derived from the `depend` clause specified in the `depen` construct that defines dependences represented by the `depend` object specified on in the `depend` clause as if the `depend` clauses of the `depen` construct were specified in the current construct.

#### If Clause [2.15][2.12]

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

- if: `[(directive-name-modifier: scalar-expression): C/C++]`
- if: `[(directive-name-modifier: scalar-logical-expression): For]`

### Map Clause [2.19.7.1][2.15.5.1]

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

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

- `map-type`: alloc, to, from, tofrom, release, delete
- `map-type-modifier`: always, close, mapper (mapper-identifier)

### Reduction Clauses [2.19.5]

- **reduce** `[reduce-modifier, ...] [reduce-identifier: list]`

Specifies a reduction among tasks.

- **reduce-identifier**: Same as for `reduction`
- **reduce-modifier**: 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** (r)

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

- **simdlen** (length)

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

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

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

- **uniform** (argument-list)

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

- **inbranch**

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

- **notinbranch**

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

### Tasking Clauses [2.10][2.9]

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

A hint to execute closely to the location of the list items.

- **allocate** (allocator: list)

See Allocate Clause, page 9 of this guide.

- **collapse** (r)

See SIMD Clauses on this page.

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

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

### Iterators

- **iterators** [2.1.6]

- **iterators-definition**: C/C++

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

- **range-specification**: begin : end [ : step ]

- **begin**, **end**: Expressions for which their types can be converted to iterator-type

- **step**: An integral expression.

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

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

### Notes
**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>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] [4.6]</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>run-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_proc_bind()</td>
<td>[6.4] [4.4]</td>
</tr>
<tr>
<td>stack-size-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>thread_limit</td>
<td>omp_get_thread_limit()</td>
<td>[6.10] [4.10]</td>
</tr>
<tr>
<td>max-active-levels-var</td>
<td>OMP_MAX_ACTIVE_LEVELS</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>omp_set_active_level()</td>
<td>omp_get_active_level()</td>
<td>---</td>
</tr>
<tr>
<td>levels-var</td>
<td>(none)</td>
<td>zero</td>
<td>omp_set_level()</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.

**OMP_ALLOCATOR** [6.21]

Sets the def-allocator-vars ICV that specifies the default allocator for allocation calls, directives and clauses that do not specify an allocator. The arg is a case-insensitive, defined allocator below (for details, see Table 2.9):  
- **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 in Table 2.9 below shows allowed allocator traits and their possible values, with the default value shown in blue.

**OMP_AFFINITY_FORMAT** [6.14]

Sets the initial value of the affinity-format-var ICV defining the format when displaying OpenMP thread affinity information. The argument is a character string that may contain as substrings one or more field specifiers, in addition to other characters. The format of each field specifier is: \%[^{[}j ]i type \. where the field type may be either the short or long names listed below (Table 5.2).

**OMP_CANCELLATION** [6.11] [4.11]

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

**OMP_DEBUG** [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** [6.15] [4.13]

Sets the default-device-var ICV that controls the default device number to use in device constructs.

**OMP_DISPLAY_AFFINITY** [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** [6.12] [4.12]

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

**OMP_DYNAMIC** [6.3] [4.3]

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

**OMP_MAX_ACTIVE_LEVELS** [6.1] [4.9]
Sets the max-active-levels-var ICV that controls the maximum number of nested active parallel regions.

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

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

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

**OMP_PLACES** [6.5] [4.7]
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** [6.4] [4.4]
Sets the value of the global bind-var ICV, setting the thread affinity policy to use for parallel regions at the corresponding nested level. policy can be the values true, false, or a comma-separated list of master, close, or spread in quotes.

**OMP_SCHEDULE** [4.14]
Sets the run-sched-var ICV for the runtime schedule kind and chunk size. modifier is one of monotonic or nonmonotonic; kind is one of static, dynamic, guided, or auto.

**OMP_STACKSIZE size[ B | K | M | G ]** [6.6] [4.7]
Sets the 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 unit is not specified, size is measured in kilobytes (k).

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

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

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

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

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

Tool Activation

Activating an OMPT Tool [4.2]
There are three steps an OpenMP implementation takes to activate a tool. This section explains how the tool and an OpenMP implementation interact to accomplish these tasks.

**Step 1. Determine whether to initialize** [4.2.2]
A tool indicates its interest in using the OMPT interface by providing a non-NULL pointer to an ompt_start_tool_result_t structure to an OpenMP implementation as a return value from ompt_start_tool.

There are three ways that a tool can provide a definition of ompt_start_tool to an OpenMP implementation:

- **Safely linking the tool’s definition of ompt_start_tool** into an OpenMP application.
- **Introducing a dynamically linked library** that includes the tool’s definition of ompt_start_tool into the application’s address space.
- **Providing the name of a dynamically linked library** appropriate for the architecture and operating system used by the application in the tool-libraries-var ICV.

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

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

- ompt_set_error
- ompt_set_never
- ompt_set_sometimes
- ompt_set_sometimes_paired
- ompt_set_always

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

All callbacks registered with ompt_set_callback or returned by ompt_get_callback use the dummy type signature ompt_callback. 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

Hold 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.

www.OpenMPCon.org

IWOMP International 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.

www.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.

www.supercalcomputing.org
www.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.

www.ukopenmpusers.co.uk