OpenMP directive syntax

A directive is a combination of the base-language mechanism and a directive-specification (the directive-name followed by optional clauses). A construct consists of a directive and, often, additional base language code.

C/C++

- `pragma omp declare reduction` (Fortran)
- `#pragma omp declare reduction (  
  | `#pragma omp declare mapper` (Fortran)
- `#pragma omp declare reduction &` (Fortran)

Fortran

- `pragma omp declare reduction`
- `#pragma omp declare reduction`

Data environment directives

threadprivate [5.2] [2.21.2]

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

C/C++

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

Fortran

- `!$omp threadprivate (list)`
- `!$omp threadprivate (list)`

declare reduction [5.5.11] [2.21.5.7]

Declares a reduction-identifier that can be used in a reduction_in_reduction, or task_reduction clause.

C/C++

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

Fortran

- `!$omp declare reduction (reduction-identifier : type-list : combiner [initializer-clause])`
- `!$omp declare reduction & (reduction-identifier : type-list : combiner [initializer-clause])`

reduction-identifier:

- `#pragma omp declare reduction (reduction-identifier : type-list : combiner [initializer-clause])`
- `!$omp declare reduction (reduction-identifier : type-list : combiner [initializer-clause])`

initialize-clause:

- `omp_initializers (initializer-expr)`

processor:

- `omp_processor (processor-ident)`

C/C++

- `omp_initializers (initializer-expr)`

processor-ident:

- `omp_processor (processor-ident)`

memory-areas:

- `omp_memory_areas (memory-area-ident)`

memory-area-ident:

- `omp_memory_areas (memory-area-ident)`

memory-management-directives:

- `omp_memory_management (memory-management-directives)`

OpenMP Examples Document

An Examples Document and a link to a GitHub repository with code samples is at [link](https://link.openmp.org/examples51).

Directives and Constructs

OpenMP constructs consist of a directive and, if defined in the syntax, an associated structured block that follows. • OpenMP directives except `simd` and any declarative directive may not appear in Fortran PURE procedures. • structured-block is a construct or block of executable statements with a single entry at the top and a single exit at the bottom. • strictly-structured-block is a structured block that isn’t strictly structured and doesn’t start with a Fortran BLOCK construct. • `omp-integer-expression` is of a C/C++ scalar int type or Fortran scalar integer type.

- `omp_logical-expression` is a C/C++ scalar expression or Fortran logical expression.

allocate [6.8] [2.13.3]

Specifies how a set of variables is allocated.

C/C++

- `#pragma omp allocate (list) [clause [ [clause ... ] ] ]`
- `#pragma omp allocate (list) [clause [ [clause ... ] ] ]`

Fortran

- `!$omp allocate (list) [clause [ [clause ... ] ] ]`
- `!$omp allocate (list) [clause [ [clause ... ] ] ]`

vendor-directives [6.7]

Specifies that OpenMP memory allocators are used for certain variables that are allocated by the associated allocate-stmt.

C/C++

- `#pragma omp allocate (list) [clause [ [clause ... ] ] ]`
- `#pragma omp allocate (list) [clause [ [clause ... ] ] ]`

Fortran

- `!$omp allocate (list) [clause [ [clause ... ] ] ]`
- `!$omp allocate (list) [clause [ [clause ... ] ] ]`

Variant directives

[begin ] [2.3.3.4]

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

C/C++

- `#pragma omp variant (variant-directive [ [variant-directive ... ] ] )`
- `#pragma omp variant (variant-directive [ [variant-directive ... ] ] )`

Fortran

- `!$omp variant (variant-directive [ [variant-directive ... ] ] )`
- `!$omp variant (variant-directive [ [variant-directive ... ] ] )`

OpenMP is a 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.
Directives and Constructs (continued)

### [begin ]declare variant (7.5.4.5) [2.3.5]
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] ...) 

#pragma omp declare variant(variant-func-id) clause ([clause] ...) ...
```

### declare simd (7.7) [2.11.5.3]
Applied to a function or subroutine to enable creation of one or more versions to process multiple arguments using SIMD instructions from a single invocation in a SIMD loop.

```plaintext
#pragma omp declare simd clause ([clause] ...) ...

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

### [begin ]declare target (7.8.1.2) [2.14.7]
A declarative directive that specifies that variables, functions, and subroutines are mapped to a device.

```plaintext
#pragma omp declare target [ ] ... 

#pragma omp declare target [ ] ...
```

### Informational and utility directives

#### requires (8.2) [2.5.1]
Specifies that an implementation must provide in order for the code to compile and to execute correctly.

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

#pragma omp requires clause ([clause] ...) ...
```

#### assert, [begin ]assumes (8.3.2.4) [2.5.2]
Provides invariants to the implementation that may be used for optimization purposes.

```plaintext
#pragma omp assumes clause ([clause] ...) ...

#pragma omp assumes clause ([clause] ...) ...
```

---

For variant-proc-name

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

#### clause-match:

```plaintext
match [context-selector-specification] 

match [context-selector-specification] ...
```

#### dispatch (7.6) [2.3.6]
Controls whether variant substitution occurs for a function call in the structured block.

```plaintext
#pragma omp dispatch clause ([clause] ...) 

#pragma omp dispatch clause ([clause] ...) ...
```

---

For device_type (host)

Identifies the target device that is associated with a device construct.

```plaintext
is_device_ptr (list) 

is_device_ptr (list)
```

---

For nocontext

Ifomp-logical-expression evaluates to true, the construct is not added to the construct set of the OpenMP context.

```plaintext
nocontext [omp-logical-expression] 

nocontext [omp-logical-expression]
```

---

For novariants

Ifomp-logical-expression evaluates to true, no variant function is selected for the call in the applicable dispatch region.

```plaintext
novariants [omp-logical-expression] 

novariants [omp-logical-expression]
```
### Directives and Constructs (continued)

#### nothing [8.4] [2.5.3]
Indicates explicitly that the intent is to have no effect.

```c
#pragma omp nothing
```

#### error [8.5] [2.5.4]
Instructs the compiler or runtime to display a message and to perform an error action.

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

#### Loop transformation constructs

**tile [9.1] [2.11.9.1]**
Tiles one or more loops.

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

**unroll [9.2] [2.11.9.2]**
Fully or partially unrolls a loop.

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

#### Parallelism constructs

**parallel [10.1] [2.6]**
Creates a team of OpenMP threads that execute the region.

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

**nowait [9.1] [2.11.9.1]**
Instructs the compiler or runtime to display a message and error.

```c
#pragma omp nowait
```

**region.**
Creates a team of OpenMP threads that execute the Parallelism constructs

**unroll** unroll [9.2] [2.11.9.2]
Fulfills or partially unrolls a loop.

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

**tile** tile [9.1] [2.11.9.1]
Tiles one or more loops.

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

**unroll**

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

#### teams [10.2] [2.7]
Creates a league of initial teams where each initial thread executes the region.

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

**work-sharing constructs**

**single** single [11.1] [2.10.2]
Applies to a loop to indicate that the loop can be transformed into a SIMD loop.

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

**scope** scope [11.2] [2.8]
Divides the execution of the enclosed structured block into separate units of work, each executed only once by one thread.

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

**teams** teams [10.2] [2.7]
Creates a league of initial teams where each initial thread executes the region.

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

### Work-distribution constructs

**single** single [11.1] [2.10.2]
Applies to a loop to indicate that the loop can be transformed into a SIMD loop.

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

**scope** scope [11.2] [2.8]
Divides the execution of the enclosed structured block into separate units of work, each executed only once by one thread.

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

**teams** teams [10.2] [2.7]
Creates a league of initial teams where each initial thread executes the region.

```c
#pragma omp teams [clause [ [], clause ] ...
```
<table>
<thead>
<tr>
<th>Directives and Constructs (continued)</th>
</tr>
</thead>
</table>

**section and sections [11.3] [2.10.1]**

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

```
#pragma omp sections [clause [...]]
| [pragma omp section]
| structured-block-sequence
| [#pragma omp section]
| structured-block-sequence
| ...
| [#pragma omp end sections]
```

**do and for [11.5.1-2] [2.11.4]**

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

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

| [pragma omp do]
| [clause [...]]
| [pragma omp end do]
```

**loop [11.7] [2.11.7]**

Specifies that the iterations of the associated loops may execute concurrently and permits the encountering thread(s) to execute the loop accordingly.

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

| [pragma omp loop]
| [clause [...]]
| [pragma omp end loop]
```

**Tasking constructs**

**task [12.5] [2.12.1]**

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

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

| [pragma omp task]
| [clause [...]]
| [pragma omp end task]
```

**Device directives and construct**

**target data [13.5] [2.14.2]**

Maps variables to a device data environment for the extent of the region.

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

| [pragma omp target data]
| [clause [...]]
| [pragma omp end target data]
```
Directives and Constructs (continued)

target enter data [13.6][2.14.3]
Maps variables to a device data environment.

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

clause:
- **depend** ([depend-modifier], dependence-type: [locator-list])
- **device** ([omp-integer-expression])
- **map** ([map-modifier, [map-modifier, ...]], [map-type]: [list])
- **nowait**

```c
#pragma omp target exit data [13.7][2.14.4]
```

clause:
- **target update** ([target-update-clause])
- **device** ([omp-integer-expression])
- **map** ([map-modifier, [map-modifier, ...]], [map-type]: [list])
- **nowait**

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

```c
#pragma omp target [structured-block]
```

```c
#pragma omp target [loosely-structured-block]
```

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

```c
#pragma omp target update [structured-block]
```

```c
#pragma omp target update [loosely-structured-block]
```

```c
#pragma omp target end taskgroup
```

```c
#pragma omp target end [controlled-block]
```

```c
#pragma omp target end [loosely-structured-block]
```

```c
#pragma omp target end [strictly-structured-block]
```

```c
#pragma omp taskwait
```

```c
#pragma omp flush
```

```c
#pragma omp depobj (depend-object)
```

Interoperability construct

interop [14.1][2.15.1]
Retrieves interoperability properties from the OpenMP implementation to enable interoperability with foreign execution contexts.

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

```c
#pragma omp interop [structured-block]
```

```c
#pragma omp interop [loosely-structured-block]
```

```c
#pragma omp interop [strictly-structured-block]
```

Synchronizations constructs

critical [15.2][2.19.1]
Restricts execution of the associated structured block to a single thread at a time.

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

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

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

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

barrier [15.3.1][2.19.2]
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
```

```c
#pragma omp barrier
```

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

```c
#pragma omp taskgroup [structured-block]
```

```c
#pragma omp taskgroup [loosely-structured-block]
```

```c
#pragma omp taskgroup [strictly-structured-block]
```

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

```c
#pragma omp taskwait [structured-block]
```

```c
#pragma omp taskwait [loosely-structured-block]
```

```c
#pragma omp taskwait [strictly-structured-block]
```

```c
flush [15.5][2.19.5]
Specifies a wait on the completion of child tasks of the current task.

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

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

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

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

```c
#pragma omp depobj [depend-object] clause
```

```
class: Any clause used for target enter data. See exception for the map clause.
```

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

```c
#pragma omp target [structured-block]
```

```c
#pragma omp target [loosely-structured-block]
```

```c
#pragma omp target [strictly-structured-block]
```

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

```c
#pragma omp target update [structured-block]
```

```c
#pragma omp target update [loosely-structured-block]
```

```c
#pragma omp target update [strictly-structured-block]
```

```c
#pragma omp taskwait
```

```c
#pragma omp flush
```

```c
#pragma omp depobj [depend-object] clause
```

```
```
atomic [15.8.4] [2.19.7] Ensures a specific storage location is accessed atomically.

C/C++

atomic [15.8.4] [2.19.7] statement

!$omp atomic clause [ [ ] clause ... ]

- or -

!$omp end atomic

clause: atomic-clause: read, write, update memory-order-clause: seq_cst, acq_rel, release, acquire, relaxed

extended-atomic: capture, compare, fail, weak

capture: Capture the value of the variable being updated atomically.

compare: Perform the atomic update conditionally.

fail (seq_cst | acquire | relaxed): Specify the memory ordering requirements for any comparison performed by any atomic conditional update that fails. Its argument overrides any other specified memory ordering.

weak: Specify that the comparison performed by a conditional atomic update may spuriously fail, evaluating to not equal even when the values are equal.

hint (hint-expression)

C/C++ statement:

if atomic clause is... statement:

read v = x;
write x = expr;
update x = x operator expr; x = expr operator x; x = intrinsic_procedure_name (x, expr-list, x)

if compare is present

if (x == y) end if

if if-clause

$omp ordered clause [ [ ] clause ... ]

structured-block

- or -

$omp ordered clause [ [ ] clause ... ]

loosely-structured-block

$omp end ordered

- or -

$omp ordered clause [ [ ] clause ... ]

doacross

ordered [15.10.2] [2.19.9] 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.

C/C++

$omp ordered clause [ [ ] clause ... ]

structured-block

if-clause:

$omp ordered clause [ [ ] clause ... ]

loosely-structured-block

$omp end ordered

- or -

$omp ordered clause [ [ ] clause ... ]

doacross

C/C++

clause (for the structured-block forms only):

threads

simplify threads or simd indicate the parallelization level with which to associate a construct.

clause (for the standalone forms only):

doacross (dependence-type : vector)

identifies cross-iteration dependences that imply additional constraints on the scheduling of loop iterations.

dependency-type:

source

specifies the satisfaction of cross-iteration dependences that arise from the current iteration. if source is specified, then the vector argument is optional; if vector is omitted, it is assumed to be omp_curIteration. at most one doacross clause can be specified on a directive with source as the dependence-type.

sink

specifies a cross-iteration dependence, where vector indicates the iteration that satisfies the dependence. if vector does not occur in the iteration space, the doacross clause is ignored. if all doacross clauses on an ordered construct are ignored then the construct is ignored.

C/C++

$omp ordered clause [ [ ] clause ... ]

loop-nest

$omp ordered clause [ [ ] clause ... ]

loop-nest

$omp end ordered clause [ nowait ]

clause: Any of the clauses accepted by the simd, for, or do directives.

$omp ordered clause [ [ ] clause ... ]

loop-nest

$omp do simd clause [ [ ] clause ... ]

loop-nest

$omp end do simd clause [ nowait ]

clause: Any of the clauses accepted by do simd.

$omp ordered clause [ [ ] clause ... ]

loop-nest

$omp ordered clause [ [ ] clause ... ]

loop-nest

$omp end ordered clause [ nowait ]

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

C/C++

$omp ordered clause [ [ ] clause ... ]

loop-nest

$omp end ordered clause [ nowait ]

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

$omp ordered clause [ [ ] clause ... ]

loop-nest

$omp ordered clause [ [ ] clause ... ]

loop-nest

$omp end ordered clause [ nowait ]

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

$omp ordered clause [ [ ] clause ... ]

loop-nest

$omp ordered clause [ [ ] clause ... ]

loop-nest

$omp end ordered clause [ nowait ]

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

C/C++

$omp ordered clause [ [ ] clause ... ]

loop-nest

$omp ordered clause [ [ ] clause ... ]

loop-nest

$omp end ordered clause [ nowait ]

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

C/C++

$omp ordered clause [ [ ] clause ... ]

loop-nest

$omp ordered clause [ [ ] clause ... ]

loop-nest

$omp end ordered clause [ nowait ]

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

C/C++

$omp ordered clause [ [ ] clause ... ]

loop-nest

$omp ordered clause [ [ ] clause ... ]

loop-nest

$omp end ordered clause [ nowait ]

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

C/C++

$omp ordered clause [ [ ] clause ... ]

loop-nest

$omp ordered clause [ [ ] clause ... ]

loop-nest

$omp end ordered clause [ nowait ]

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

C/C++

$omp ordered clause [ [ ] clause ... ]

loop-nest

$omp ordered clause [ [ ] clause ... ]

loop-nest

$omp end ordered clause [ nowait ]

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

C/C++

$omp ordered clause [ [ ] clause ... ]

loop-nest

$omp ordered clause [ [ ] clause ... ]

loop-nest

$omp end ordered clause [ nowait ]

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

C/C++

$omp ordered clause [ [ ] clause ... ]

loop-nest

$omp ordered clause [ [ ] clause ... ]

loop-nest

$omp end ordered clause [ nowait ]

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

C/C++

$omp ordered clause [ [ ] clause ... ]

loop-nest

$omp ordered clause [ [ ] clause ... ]

loop-nest

$omp end ordered clause [ nowait ]

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

C/C++

$omp ordered clause [ [ ] clause ... ]

loop-nest

$omp ordered clause [ [ ] clause ... ]

loop-nest

$omp end ordered clause [ nowait ]

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

C/C++

$omp ordered clause [ [ ] clause ... ]

loop-nest

$omp ordered clause [ [ ] clause ... ]

loop-nest

$omp end ordered clause [ nowait ]

clause: Any of the clauses accepted by do ordered or do simd.
**Directives and Constructs (continued)**

**distribute parallel do simd and distribute parallel for simd [17]** [2.11.6.4]
Specifies that a loop can be executed concurrently using SIMD instructions in parallel by multiple threads that are members of multiple teams.

```fortran
!$omp distributing parallel for simd \ [clause[ ]... ]
loop-nest
```

```c/c++
#pragma omp distribute parallel for simd[ ]
```

Clause: Any clause used for distribute, parallel for, simd, or parallel do simd.

**taskloop simd [17]** [2.12.3]
Specifies that a loop can be executed concurrently using SIMD instructions, and that those iterations will also be executed in parallel using OpenMP tasks.

```fortran
!$omp taskloop simd [clause[ ]... ]
loop-nest
```

```c/c++
#pragma omp taskloop simd[ ]
```

Clause: Any clause used for taskloop, simd, or parallel do simd.

**parallel do and parallel for [17]** [2.16.1]
Specifies a parallel construct containing a worksharing-loop construct with a canonical loop nest and no other statements.

```fortran
!$omp parallel for [clause[ ]... ]
loop-nest
```

```c/c++
#pragma omp parallel for [ ]
```

Clause: Any clause used for parallel or do simd.

**parallel loop [17]** [2.16.2]
Shortcut for specifying a parallel construct containing a loop construct with a canonical loop nest and no other statements.

```fortran
!$omp parallel loop [clause[ ]... ]
loop-nest
```

```c/c++
#pragma omp parallel loop[ ]
```

Clause: Any clause used for parallel or loop.

**parallel sections [17]** [2.16.3]
Shortcut for specifying a parallel construct containing a sections construct and no other statements.

```fortran
!$omp parallel sections [clause[ ]... ]
{ [ ] } loop-nest
```

```c/c++
#pragma omp parallel sections[ ]
```

Clause: Any clause used for parallel or sections [except the nowait clause].

**parallel workshare [17]** [2.16.4]
Shortcut for specifying a parallel construct containing a workshare construct and no other statements.

```fortran
!$omp workshare[ ]
```

```c/c++
#pragma omp workshare[ ]
```

Clause: Any clause used for parallel or workshare.

**parallel masked taskloop simd [17]** [2.16.10]
Shortcut for specifying a parallel construct containing a masked taskloop simd construct and no other statements.

```fortran
!$omp masked taskloop simd[ ]
```

```c/c++
#pragma omp masked taskloop simd[ ]
```

Clause: Any clause used for masked taskloop simd or parallel except the in_reduction clause.

**teams distribute [17]** [2.16.11]
Shortcut for specifying a teams construct containing a distribute construct and no other statements.

```fortran
!$omp teams distribute[ ]
```

```c/c++
#pragma omp teams distribute[ ]
```

Clause: Any clause used for teams or distribute.

**teams distribute simd [17]** [2.16.12]
Shortcut for specifying a teams construct containing a distribute simd construct and no other statements.

```fortran
!$omp teams distribute simd[ ]
```

```c/c++
#pragma omp teams distribute simd[ ]
```

Clause: Any clause used for teams or distribute simd.

**teams distribute parallel do and teams distribute parallel for [17]** [2.16.13]
Shortcut for specifying a teams construct containing a distribute parallel worksharing-loop construct and no other statements.

```fortran
!$omp teams distribute parallel do[ ]
```

```c/c++
#pragma omp teams distribute parallel do[ ]
```

Clause: Any clause used for teams, distribute parallel for, or distribute parallel do.

**teams distribute parallel do simd and teams distribute parallel for simd [17]** [2.16.14]
Shortcut for specifying a teams construct containing a distribute parallel simd or distribute parallel do simd construct and no other statements.

```fortran
!$omp teams distribute parallel do simd[ ]
```

```c/c++
#pragma omp teams distribute parallel do simd[ ]
```

Clause: Any clause used for teams, distribute parallel do simd, or distribute parallel do simd.

**teams loop [17]** [2.16.15]
Shortcut for specifying a teams construct containing a loop construct and no other statements.

```fortran
!$omp teams loop[ ]
```

```c/c++
#pragma omp teams loop[ ]
```

Clause: Any clause used for teams or loop.
### Directives and Constructs (continued)

**target parallel** [17] [2.16.16]
Shortcut for specifying a target construct containing a parallel construct and no other statements.

- **C/C++**
  - `#pragma omp target parallel [clause[,clause]... ] loop-nest`
- **Fortran**
  - `!$omp target parallel [clause[,clause]... ]
  - `& !$omp end target parallel`
  - `loop-nest`

*clause*: Clauses used for target or parallel except for copyin.

**target parallel do and target parallel for** [17] [2.16.17]
Shortcut for specifying a target construct with a parallel working-loop construct and no other statements.

- **C/C++**
  - `#pragma omp target parallel for [clause[,clause]... ] loop-nest`
- **Fortran**
  - `& !$omp target parallel do [clause[,clause]... ]
  - `& !$omp end target parallel do`
  - `loop-nest`

**target parallel do simd and target parallel for simd** [17] [2.16.18]
Shortcut for specifying a target construct with a parallel working-loop SIMD construct and no other statements.

- **C/C++**
  - `#pragma omp target parallel for simd [clause[,clause]... ] loop-nest`
- **Fortran**
  - `& !$omp target parallel do simd [clause[,clause]... ]
  - `& !$omp end target parallel do simd`
  - `loop-nest`

**target parallel loop** [17] [2.16.19]
Shortcut for specifying a target construct containing a parallel loop construct and no other statements.

- **C/C++**
  - `#pragma omp target parallel loop [clause[,clause]... ] loop-nest`
- **Fortran**
  - `& !$omp target parallel loop [clause[,clause]... ]
  - `& !$omp end target parallel loop`
  - `loop-nest`

**target simd** [17] [2.16.20]
Shortcut for specifying a target construct containing a simd construct and no other statements.

- **C/C++**
  - `#pragma omp target simd [clause[,clause]... ] loop-nest`
- **Fortran**
  - `& !$omp target simd [clause[,clause]... ]
  - `& !$omp end target simd`

*clause*: Any clause used for target or simd.

**target teams** [17] [2.16.21]
Shortcut for specifying a target construct containing a teams construct and no other statements.

- **C/C++**
  - `#pragma omp target teams [clause[,clause]... ] loop-nest`
- **Fortran**
  - `& !$omp target teams [clause[,clause]... ]
  - `& !$omp end target teams`

*clause*: Any clause used for target or teams.

**target teams distribute** [17] [2.16.22]
Shortcut for specifying a target construct containing a teams distribute construct and no other statements.

- **C/C++**
  - `#pragma omp target teams distribute [clause[,clause]... ] loop-nest`
- **Fortran**
  - `& !$omp target teams distribute [clause[,clause]... ]
  - `& !$omp end target teams distribute`

*clause*: Any clause used for target or teams distribute.

**target teams loop** [17] [2.16.24]
Shortcut for specifying a target construct containing a teams loop construct and no other statements.

- **C/C++**
  - `#pragma omp target teams loop [clause[,clause]... ] loop-nest`
- **Fortran**
  - `& !$omp target teams loop [clause[,clause]... ]
  - `& !$omp end target teams loop`

*clause*: Any clause used for target or teams loop.

**target teams distribute parallel do** and **target teams distribute parallel for** [17] [2.16.25]
Shortcut for specifying a target construct containing teams distribute parallel for, teams distribute parallel do and no other statements.

- **C/C++**
  - `#pragma omp target teams distribute parallel for do [clause[,clause]... ] loop-nest`
- **Fortran**
  - `& !$omp target teams distribute parallel do [clause[,clause]... ]
  - `& !$omp end target teams distribute parallel do`

*clause*: Any clause used for target, teams distribute parallel for, or teams distribute parallel do.

**target teams distribute parallel do simd** and **target teams distribute parallel for simd** [17] [2.16.26]
Shortcut for specifying a target construct containing teams distribute parallel worksharing-loop SIMD construct and no other statements.

- **C/C++**
  - `#pragma omp target teams distribute parallel for simd [clause[,clause]... ] loop-nest`
- **Fortran**
  - `& !$omp target teams distribute parallel for simd [clause[,clause]... ]
  - `& !$omp end target teams distribute parallel for simd`

*clause*: Any clause used for target, teams distribute parallel for, or teams distribute parallel do simd.

**Notes**

---

© 2023 OpenMP ARB

www.openmp.org
Data sharing attribute clauses [5.4] [2.21.4]
Additional data sharing attribute clauses are _is_device_ptr, use_device_ptr, has_device_addr, and use_device_addr. These clauses are described at the directives that accept them.

default [shared | firstprivate | private | none]
Default data-sharing attributes are disabled. All variables in a construct must be declared inside the construct or appear in a data-sharing attribute clause.

allocate clause [6.6] [2.13.4]
allocate ([allocator | ] list)
allocate (allocate modifier, [allocate modifier | ] list)
allocate-modifier
allocator (allocator)
allocation is an expression of:
C++: type omp_allocate_handle_t
Fortran: kind omp_allocate_handle_t
alignment (alignment)
alignment: A constant positive integer power of 2.

if clause [3.4] [2.18]
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 no modifier 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: ] omp-logical-expression)
if (begin)
if (end)
if (monitor)
if (nowait)

order clause [10.3] [2.21.3]
order (order-modifier | ) concurrent
order-modifier: reproducible, unconstrained

nowait clause [15.6]
nowait
Overrides any synchronization that would otherwise occur at the end of a construct. It can also specify that an interoperability requirement set includes the nowait property. If the construct includes an implicit barrier, the nowait clause specifies that the barrier will not occur.

reduction clause [5.5.8] [2.21.5.4]
reduction (reduction-modifier, ] reduction-identifier | list)
reduction-modifier: specified a reduction-identifier and one or more list items.
reduction-modifier: incan, task, default C= reduction-identifier
Either an id-expression or one of the following operators: +, *, |, &, <, >, <=, >=
c reduction-identifier: Either an identifier or one of the following operators:
+ | * | & | <, <=, >=
For reduction-identifier:
Either a base language identifier, a user-defined operator, one of the following operators: +, *, |, &,
or one of the following intrinsic procedure names: max, min, and, or, leq, neqv.

device clause [13.2]
device (ancestor | device_num | ] device-description)
Identifies the target device that is associated with a device construct.
device-description: An expression of type integer that refers to the device number or, if ancestor modifier is specified, must be 1.

iterator [3.2.6] [2.1.6]
Identifiers that expand to multiple values in the clause on which they appear.
iterator (iterators-definition)
iterators-definition: iterator-specifier | iterators-definition
iterators-specifier:
iterator-type | identifier = range-specification identifier:
A base language identifier.
range-specification: begin | end | step
begin, end: Expressions for which their types can be converted to iterator-type step:
An integral expression.
iterator-type: [C|C++] A type name. For A type specifier.
# Runtime Library Routines

## Thread team routines

### `omp_set_num_threads` [18.2.1] [3.2.1]
Affects the number of threads used for subsequent parallel constructs not specifying a `num_threads` clause, by setting the value of the first element of the `nthreads-var` ICV of the current task to `num_threads`.

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

### `omp_get_num_threads` [18.2.2] [3.2.2]
Returns the number of threads in the current team. The binding region for an `omp_get_num_threads` region is the innermost enclosing parallel region. If called from the sequential part of a program, this routine returns 1.

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

### `omp_get_max_threads` [18.2.3] [3.2.3]
Returns an upper bound on the number of threads that could be used to form a new team if a parallel construct without a `num_threads` clause were encountered after execution returns from this routine.

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

### `omp_get_thread_num` [18.2.4] [3.2.4]
Returns the thread number of the calling thread, within the current team.

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

### `omp_in_parallel` [18.2.5] [3.2.5]
Returns true if the active-levels-var ICV is greater than zero; otherwise it returns false.

```c
logical function omp_in_parallel (void);
```

### `omp_set_dynamic` [18.2.6] [3.2.6]
Enables or disables dynamic adjustment of the number of threads available for the execution of subsequent parallel regions by setting the value of the `dyn-var` ICV.

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

### `omp_get_dynamic` [18.2.7] [3.2.7]
Returns true if dynamic adjustment of the number of threads is enabled for the current task. ICV: `dyn-var`

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

### `omp_get_cancellation` [18.2.8] [3.2.8]
Returns true if cancellation is enabled; otherwise it returns false. ICV: `cancel-var`

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

### `omp_set_schedule` [18.2.11] [3.2.11]
Affects the schedule that is applied when runtime is used as schedule kind, by setting the value of the `sched-var` ICV.

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

### `omp_get_schedule` [18.2.12] [3.2.12]
Returns the schedule applied when runtime schedule is used. ICV: `sched-var`

```c
void get_schedule (omp_sched_t *kind, int *chunk_size);
```

### `omp_get_max_active_levels` [18.2.13] [3.2.13]
Returns the maximum number of nested active parallel regions when a new nested parallel region is generated by the current task, by setting `max-active-levels-var` ICV.

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

### `omp_get_active_level` [18.2.14] [3.2.14]
Returns the number of active levels of parallelism supported.

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

### `omp_set_max_active_levels` [18.2.15] [3.2.15]
Limits the number of nested active parallel regions when a new nested parallel region is generated by the current task, by setting `max-active-levels-var` ICV.

```c
void omp_set_max_active_levels (int max_levels);
```

### `omp_get_place_num` [18.3.5] [3.3.5]
Returns numerical identifiers of the processors available to the execution environment in the specified place.

```c
int omp_get_place_num (void);
```

### `omp_get_place_num_procs` [18.3.6] [3.3.6]
Returns the number of processors available to the execution environment in the specified place.

```c
int omp_get_place_num_procs (int place_num);
```

### `omp_get_ancestor_thread_num` [18.2.18] [3.2.18]
Returns, for a given nested level of the current thread, the thread number of the ancestor of the current thread.

```c
int omp_get_ancestor_thread_num (int level);
```

### `omp_get_team_size` [18.2.19] [3.2.19]
Returns, for a given nested level of the current thread, the size of the thread team to which the ancestor or the current thread belongs.

```c
int omp_get_team_size (int level);
```

### `omp_get_active_level` [18.2.20] [3.2.20]
Returns the number of active, nested parallel regions on the device enclosing the task containing the call. ICV: `active-level-var`

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

### `omp_get_place_active_level` [18.3.2] [3.3.2]
Returns the number of active levels of parallelism supported.

```c
int omp_get_place_active_level (void);
```

### `omp_set_place_active_level` [18.3.3] [3.3.3]
Limits the number of active parallel regions on the device enclosing the task containing the call.

```c
void omp_set_place_active_level (int place_num, int level);
```

### `omp_get_num_places` [18.3.4] [3.3.4]
Returns numerical identifiers of the processors available to the execution environment in the specified place.

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

### `omp_get_proc_bind` [18.3.1] [3.3.1]
Returns the thread affinity policy to be used for the subsequent nested parallel regions that do not specify a `proc_bind` clause.

```c
int omp_get_proc_bind (void);
```

### `omp_set_proc_bind` [18.3.12] [3.3.12]
Limits the number of active parallel regions when a new nested parallel region is generated by the current task, by setting `max-active-levels-var` ICV.

```c
void omp_set_proc_bind (int proc_bind_kind);
```

### `omp_get_place_proc_ids` [18.3.5] [3.3.5]
Returns the number of places available to the execution environment in the place list.

```c
int omp_get_place_proc_ids (int place_num);
```

### `omp_get_proc_bind` [18.3.12] [3.3.12]
Limits the number of active parallel regions when a new nested parallel region is generated by the current task, by setting `max-active-levels-var` ICV.

```c
void omp_set_proc_bind (int proc_bind_kind);
```

### `omp_get_place_num_procs` [18.3.6] [3.3.6]
Returns the number of processors available to the execution environment in the specified place.

```c
int omp_get_place_num_procs (int place_num);
```

### `omp_get_proc_bind` [18.3.12] [3.3.12]
Limits the number of active parallel regions when a new nested parallel region is generated by the current task, by setting `max-active-levels-var` ICV.

```c
void omp_set_proc_bind (int proc_bind_kind);
```

### `omp_get_place_active_level` [18.3.2] [3.3.2]
Returns the number of active levels of parallelism supported.

```c
int omp_get_place_active_level (void);
```

### `omp_get_place_num_procs` [18.3.6] [3.3.6]
Returns the number of processors available to the execution environment in the specified place.

```c
int omp_get_place_num_procs (int place_num);
```

### `omp_get_proc_bind` [18.3.12] [3.3.12]
Limits the number of active parallel regions when a new nested parallel region is generated by the current task, by setting `max-active-levels-var` ICV.

```c
void omp_set_proc_bind (int proc_bind_kind);
```
OpenMP API 5.2

Runtime Library Routines (continued)

**omp_get_partition_num_places [18.3.6] [3.3.6]**
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_place_nums [18.3.7] [3.3.7]**
Returns the list of place numbers corresponding to the places in the place-partition-var ICV of the innermost implicit task.

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

**omp_set_affinity_format [18.3.8] [3.3.8]**
Sets the affinity format to be used on the device by setting the value of the affinity-format-var ICV.

```c
void omp_set_affinity_format (const char *format);
```

**omp_get_affinity_format [18.3.9] [3.3.9]**
Returns the value of the affinity-format-var ICV on the device.

```c
size_t omp_get_affinity_format (char *buffer, size_t buffer_size);
```

**omp_capture_affinity [18.3.10] [3.3.10]**
Prints the OpenMP thread affinity information using the format specification provided.

```c
void omp_capture_affinity (const char *format);
```

**omp_display_affinity [18.3.11] [3.3.11]**
Prints the OpenMP thread affinity information into a buffer using the format specification provided.

```c
void omp_display_affinity (const char *format);
```

**Tasking routines**

**omp_get_max_task_priority [18.5.1] [3.5.1]**
Returns the maximum value that can be specified in the priority clause.

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

**omp_is_initial_device [18.5.3] [3.5.3]**
Returns true if the current task is executing on the host device; otherwise, it returns false.

```c
int omp_is_initial_device (void);
```

**Resource relinquishing routines**

**omp_pause_resource [18.6.1] [3.6.1]**
Suspends execution for the duration of the OpenMP region.

```c
void omp_pause_resource (const char *resource);
```

**omp_pause_resource_all [18.6.2] [3.6.2]**
Suspends execution for the duration of the OpenMP region for all threads.

```c
void omp_pause_resource_all (const char *resource);
```

**Device information routines**

**omp_get_num_places [18.7.1] [3.7.1]**
Returns the number of processors that are available to the device at the time the routine is called.

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

**omp_set_default_device [18.7.2] [3.7.2]**
Assigns the value of the default-device-var ICV, which determines the default target device.

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

**omp_get_num_devices [18.7.4] [3.7.4]**
Returns the number of non-host devices available for offloading code or data.

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

**omp_get_device_num [18.7.5] [3.7.5]**
Returns the device number of the device on which the calling thread is executing.

```c
int omp_get_device_num (void);
```

**Device memory routines**

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

**omp_target_alloc [18.8.1] [3.8.1]**
Allocates memory in a device data environment and returns a device pointer to that memory.

```c
type(c_ptr) function omp_target_alloc ( & device_num, bind(c) use, intrinsic :: iso_c_binding, only : c_ptr, & c_size_t c_int, integer(c_size_t) size, integer(c_int) value : device_num);
```

**omp_target_free [18.8.2] [3.8.2]**
Frees the device memory allocated by the omp_target_alloc routine.

```c
void *omp_target_free (void *device_ptr, int device_num);
```
**Runtime Library Routines (continued)**

**omp_target_is_present** [18.8.3] [3.8.3]
Tests whether a host pointer refers to storage that is mapped to a given device.

```
int omp_target_is_present (const void *ptr, int device_num);
```

**omp_target_accessible** [18.8.4] [3.8.4]
Tests whether host memory is accessible from a given device.

```
int omp_target_is_accessible (const void *ptr, size_t_t size, int device_num);
```

**omp_target_memcpy** [18.8.5] [3.8.5]
Copies memory between any combination of host and device pointers.

```
void omp_target_memcpy (void *dst, const void *src, size_t_t volume, const size_t_t *size, const size_t_t *src_offset, size_t_t *dst_offset, int dev_num, int depo_count, omp_depend_t *depo_list);
```

**omp_target_memcpy_async** [18.8.7] [3.8.7]
Performs a copy between any combination of host and device pointers asynchronously.

```
void omp_target_memcpy_async (void *dst, const void *src, size_t_t volume, const size_t_t *size, const size_t_t *src_offset, size_t_t *dst_offset, int dev_num, const size_t_t *src_device_num, int depo_count, int depo_depend_kind, optional depo_list_t *depo_list);
```

**omp_target_memcpy_rect_async** [18.8.8] [3.8.8]
Asynchronously performs a copy between any combination of host and device pointers.

```
void omp_target_memcpy_rect_async (void *dst, const void *src, size_t_t volume, const size_t_t *size, const size_t_t *src_offset, size_t_t *dst_offset, int dev_num, int depo_count, int depo_depend_kind, optional depo_list_t *depo_list);
```

**omp_target_disassociate_ptr** [18.8.10] [3.8.10]
Removes the association between a host pointer and a device address on a given device.

```
void omp_target_disassociate_ptr (const void *ptr, int device_num);
```

**omp_targetမget_ptr** [18.8.11] [3.8.11]
Returns the device pointer that is associated with a host pointer for a given device.

```
void *omp_target_get_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** [18.9.1] [3.9.1]

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

**Initialize lock with hint** [18.9.2] [3.9.2]

```
void omp_init_lock_with_hint (omp_lock_t *lock);
void omp_init_nest_lock_with_hint (omp_nest_lock_t *nlock);
```

**Destroy lock** [18.9.3] [3.9.3]
Ensures that the OpenMP lock is uninitialized.

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

**Set lock** [18.9.4] [3.9.4]
Sets an OpenMP lock. The calling task region is suspended until the lock is set.

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

**Unset lock** [18.9.5] [3.9.5]

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

---

© 2023 OpenMP ARB  OMP1223
www.openmp.org
Runtime Library Routines (continued)

Test lock [18.9.6] [3.9.6]
Attempt to set an OpenMP lock but do not suspend execution of the task executing the routine.

```c
int omp_test_lock (omp_lock_t *lock);
int omp_test_nested_lock (omp_nested_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.

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

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

```c
void omp_fullfill_event (event);
void omp_destroy_event (event);
```

Interoperability routines
Interoperability routines

```c
omp_get_num_interop_properties [18.12.1] [3.12.1]
Retrieves the number of implementation-defined properties available for an omp_interop_t object.

```c
int omp_get_num_interop_properties (omp_interop_t *interop);
```

```c
omp_get_interop_int [18.12.2] [3.12.2]
Retrieves an integer property from an omp_interop_t object.

```c
omp_intptr_t omp_get_interop_int (const omp_interop_t *interop, omp_interop_property_t property_id, int *ret_code);
```

```c
omp_get_interop_ptr [18.12.3] [3.12.3]
Retrieves a pointer property from an omp_interop_t object.

```c
void *omp_get_interop_ptr (const omp_interop_t *interop, omp_interop_property_t property_id, int *ret_code);
```

```c
omp_get_interop_str [18.12.4] [3.12.4]
Retrieves a string property from an omp_interop_t object.

```c
const char *omp_get_interop_str (const omp_interop_t *interop, omp_interop_property_t property_id, int *ret_code);
```

Memory management routines
Memory Management Types [18.13.1] [3.13.1]
The omp_alloctrait_t struct in C/C++ and the omp_alloctrait type in Fortran define members named key and value, with these types and values:

```c
enum omp_alloctrait_key_t
For integer omp_alloctrait_key_kind
omp_atk_x where X may be one of sync_hint, alignment, access, pool_size, fallback, fb_data, pinned, partition
```

```c
enum omp_alloctrait_value_t
For integer omp_alloctrait_val_kind
numtraits, traits, trait(int), traits(*)
```

```c
void omp_malloc (size_t size, void * * allocator);
void omp_aligned_malloc (size_t size, void * * allocator, integer(omp_allocator_handle_kind) allocator);
```

```c
void omp_calloc (size_t nmemb, size_t size, void * * allocator);
void omp_aligned_calloc (size_t nmemb, size_t size, void * * allocator, integer(omp_allocator_handle_kind) allocator);
```

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

omp_free [18.13.7] [3.13.7]
Dealocates previously allocated memory.

```c
void omp_destroy_allocator (void);
```

```c
type(c_ptr) function omp_atak (size_t size, allocator bind(c) use, intrinsic :: iso_c_binding, only : c_ptr, c_size_t integer(c_size_t), value :: size integer(omp_allocator_handle_kind), value :: allocator);
```

```c
type(c_ptr) function omp_aligned_alloc (& alignment, size, allocator bind(c) use, intrinsic :: iso_c_binding, only : c_ptr, c_size_t integer(c_size_t), value :: alignment, size integer(omp_allocator_handle_kind), value :: allocator);
```

omp_calloc and omp_aligned_calloc [18.13.8] [3.13.8]
Request a zero-initialized memory allocation from a memory allocator.

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

```c
void omp_aligned_alloc (size_t size, void * * allocator, integer(omp_allocator_handle_kind) allocator);
```

```c
void omp_set_default_allocator (void *ptr, void * * allocator = omp_null_allocator);
```

OMP1223
### Runtime Library Routines (continued)

#### omp_realloc

Reallocates the given area of memory originally allocated by `free_allocator` using `allocor`, moving and resizing if necessary.

```c
void *omp_realloc(void *ptr, size_t size, 
    omp_allocator_handle_t allocator, 
    omp allocator_handle_t free_allocator);
```

#### omp_control_tool

Enables a program to pass commands to an active tool.

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

**integer function omp_control_tool**

- **command**: `omp_control_tool_start`
- **modifier**: `true`
- **return**: `null`

#### Environment display routine

#### omp_display_env

Displays the OpenMP version number and the values of ICVs associated with environment variables.

```c
void omp_display_env (int verbose);
```

**subroutine omp_display_env**

- **verbose**: `true`
- **return**: `null`

### Environment Variables

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

**OMP_AFFINITY_FORMAT** [21.2.5] [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 value is case-sensitive, and leading and trailing whitespace is significant. The format of each field specifier is: `%\[

#### Allocators

- **Sync**: contended, uncontented, serialized, private
- **Alignment**: 1 byte; Positive integer value that is a power of 2
- **Access**: all, group, thread
- **Pool Size**: Positive integer value (default is impl. defined)
- **Fallback**: default_mem_fb, null fb, abort_fb, allocator_fb
- **Data**: An allocator handle (No default)
- **Pinned**: true, false
- **Partition**: environment, nearest, blocked, interleaved

**Predefined allocators, memory space, and trait values** [6.2.9]

- **Default**: `omp_default_mem_alloc`
- **Large Cap Memory**: `omp_large_cap_mem_alloc`
- **Const Memory**: `omp_const_mem_alloc`
- **High BW Memory**: `omp_high_bw_mem_alloc`
- **Low Lat Memory**: `omp_low_lat_mem_alloc`
- **Group Memory**: `omp_group_mem_alloc`
- **Pthread Memory**: `omp pthread_mem_alloc`
- **Thread Memory**: `omp_thread_mem_alloc`

#### Examples

```c
setenv OMP_ALLOCATOR omp_high_bw_mem_alloc
setenv OMP_ALLOCATOR
    omp large_cap_mem_space : alignment=16, \n    pinned=true
```

### Memory space names

- **Default Memory**: `omp default mem_space`
- **Large Cap Memory**: `omp large cap mem_space`
- **Const Memory**: `omp const mem_space`
- **High BW Memory**: `omp high bw mem_space`
- **Low Lat Memory**: `omp low lat mem_space`
- **Group Memory**: `omp group mem alloc`
- **Pthread Memory**: `omp pthread mem alloc`
- **Thread Memory**: `omp thread mem alloc`

#### Environment display routine

**OMP_DISPLAY_AFFINITY** [21.2.4] [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. `var` may be `true` or `false`.

**OMP DISPLAY_ENV** [21.7] [6.12]

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

**OMP_DYNAMIC** [21.1.1] [6.3]

Sets the initial value of 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.

**OMP_MAX_ACTIVE_LEVELS** [21.1.4] [6.6]

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

**OMP_MAX_TASK_PRIORITY** [21.2.8] [6.16]

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

**OMP_NUM_TEAMS** [21.6.1] [6.23]

Sets the maximum number of teams created by a team construct by setting the teams-var ICV.

**OMP_NUM_THREADS** [21.1.2] [6.2]

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

**OMP PLACES** [21.1.6] [6.5]

Sets the initial value of the place-partition-var ICV that defines the OpenMP places available to the execution environment. places is an abstract name (threads, cores, sockets, ll_caches, numa_domains) or an ordered list of places where each place of brace-delimited numbers is an unordered set of processors on a device.
### Environment Variables

**OMP_PROC_BIND policy** [21.1.7] [6.4]  
Sets the initial value of the global bind-var ICV, setting the thread affinity policy to use for parallel regions at the corresponding nested level. *policy* can have the values `true`, `false`, or a comma-separated list of `primary`, `close`, or `spread` in quotes.

**OMP_SCHEDULE [modifier:kind, chunk]** [21.2.1] [6.1]  
Sets the run-sched-var ICV for the runtime schedule kind and chunk size. *modifier* is one of `monotonic` or `nonmonotonic`; *kind* is one of `static`, `dynamic`, `guided`, or `auto`.

**OMP_STACKSIZE size B | K | M | G** [21.2.2] [6.6]  
Sets the stacksize-var ICV that specifies the size of the stack for threads created by the OpenMP implementation. *size* is a positive integer that specifies stack size. *B* is bytes, *K* is kilobytes, *M* is megarabytes, and *G* is gigabytes. If unit is not specified, *size* is in units of *K*.

**OMP_TARGET_OFFLOAD** [21.2.8] [6.17]  
Sets the initial value of the target-offload-var ICV. The value must be one of `mandatory`, `disabled`, or `default`.

**OMP_TEAMS_THREAD_LIMIT** [21.6.2] [6.24]  
Sets the maximum number of OpenMP threads to use in each contention group created by a teams construct by setting the teams-thread-limit-var ICV.

**OMP_THREAD_LIMIT** [21.1.3] [6.10]  
Sets the maximum number of OpenMP threads to use in a contention group by setting the thread-limit-var ICV.

**OMP_TOOL** (enabled) [21.3.1] [6.18]  
Sets the tool-var ICV. If disabled, no first-party tool will be activated. If enabled, the OpenMP implementation will try to find and activate a first-party tool.

**OMP_TOOL_LIBRARIES** [21.3.2] [6.19]  
Sets the tool-libraries-var ICV to a list of tool libraries that will be considered for use on a device where an OpenMP implementation is being initialized. *library-list* is a space-separated list of dynamically-linked libraries, each specified by an absolute path.

**OMP_TOOL_VERBOSE_INIT** [21.3.3] [6.20]  
Sets the tool-verbose-init-var ICV, which controls whether an OpenMP implementation will verbosely log the registration of a tool. The value must be a filename or one of `disabled`, `stdout`, or `stderr`.

**OMP_WAIT_POLICY** [21.2.3] [6.7]  
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`. Default is implementation defined.

---

### 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 host and target devices are modified accordingly. Certain environment variables may be extended with device-specific environment variables with the following syntax: `<ENV_VAR>_DEV[/<device_num>]`. Device-specific environment variables must not correspond to environment variables that initialize ICVs with the global scope.

#### Table of ICV Initial Values, Ways to Modify and to Retrieve ICV Values, and Scope

<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>Scope</th>
<th>Env. Var. Ref.</th>
</tr>
</thead>
<tbody>
<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>Device</td>
<td>![21.2.5] [6.14]</td>
</tr>
<tr>
<td>bind-var</td>
<td>OMP_PROC_BIND</td>
<td>Implementation defined.</td>
<td>omp_set_proc_bind()</td>
<td>omp_get_proc_bind()</td>
<td>Data env.</td>
<td>![21.1.7] [6.4]</td>
</tr>
<tr>
<td>cancel-var</td>
<td>OMP_CANCELLATION</td>
<td>false</td>
<td>(none)</td>
<td>omp_get_cancellation()</td>
<td>Global</td>
<td>![21.2.6] [6.11]</td>
</tr>
<tr>
<td>debug-var</td>
<td>OMP_DEBUG</td>
<td>disabled</td>
<td>(none)</td>
<td>(none)</td>
<td>Global</td>
<td>![21.4.1] [6.21]</td>
</tr>
<tr>
<td>def-allocate-var</td>
<td>OMP_ALLOCATOR</td>
<td>Implementation defined.</td>
<td>omp_set_default_allocator()</td>
<td>omp_get_default_allocator()</td>
<td>Impl. Task</td>
<td>![21.5.1] [6.12]</td>
</tr>
<tr>
<td>def-sched-var</td>
<td>(none)</td>
<td>Implementation defined.</td>
<td>(none)</td>
<td>(none)</td>
<td>Device</td>
<td>--</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>Data env.</td>
<td>![21.2.7] [6.15]</td>
</tr>
<tr>
<td>display-affinity-var</td>
<td>OMP_DISPLAY_AFFINITY</td>
<td>false</td>
<td>(none)</td>
<td>(none)</td>
<td>Global</td>
<td>![21.2.4] [6.13]</td>
</tr>
<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>Data env.</td>
<td>![21.11.3] [6.3]</td>
</tr>
<tr>
<td>explicit-task-var</td>
<td>(none)</td>
<td>false</td>
<td>(none)</td>
<td>omp_in_explicit_task()</td>
<td>Data env.</td>
<td>--</td>
</tr>
<tr>
<td>final-task-var</td>
<td>(none)</td>
<td>false</td>
<td>(none)</td>
<td>omp_in_final()</td>
<td>Data env.</td>
<td>--</td>
</tr>
<tr>
<td>levels-var</td>
<td>(none)</td>
<td>zero</td>
<td>(none)</td>
<td>omp_get_level()</td>
<td>Data env.</td>
<td>--</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>Global</td>
<td>![21.2.9] [6.16]</td>
</tr>
<tr>
<td>nteams-var</td>
<td>OMP_NUM_TEAMS</td>
<td>zero</td>
<td>omp_set_num_teams()</td>
<td>omp_get_num_teams()</td>
<td>Device</td>
<td>![21.6.1] [6.23]</td>
</tr>
<tr>
<td>nthreads-var</td>
<td>OMP_NUM_THREADS</td>
<td>Implementation defined.</td>
<td>omp_set_num_threads()</td>
<td>omp_get_num_threads()</td>
<td>Data env.</td>
<td>![21.1.2] [6.2]</td>
</tr>
<tr>
<td>num-procs-var</td>
<td>(none)</td>
<td>Implementation defined.</td>
<td>(none)</td>
<td>omp_get_num_procs()</td>
<td>Device</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>Impl. Task</td>
<td>![21.1.6] [6.5]</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>Data env.</td>
<td>![21.2.1] [6.1]</td>
</tr>
<tr>
<td>stacksize-var</td>
<td>OMP_STACKSIZE</td>
<td>Implementation defined.</td>
<td>(none)</td>
<td>(none)</td>
<td>Device</td>
<td>![21.2.2] [6.6]</td>
</tr>
<tr>
<td>target-offload-var</td>
<td>OMP_TARGET_OFFLOAD</td>
<td>DEFAULT</td>
<td>(none)</td>
<td>(none)</td>
<td>Global</td>
<td>![21.2.8] [6.17]</td>
</tr>
<tr>
<td>team-size-var</td>
<td>(none)</td>
<td>one</td>
<td>(none)</td>
<td>omp_get_num_teams()</td>
<td>Team</td>
<td>--</td>
</tr>
<tr>
<td>teams-thread-limit-var</td>
<td>OMP_TEAMS_THREAD_LIMIT</td>
<td>zero</td>
<td>omp_set_teams_thread_limit()</td>
<td>omp_get_teams_thread_limit()</td>
<td>Device</td>
<td>![21.6.2] [6.24]</td>
</tr>
<tr>
<td>thread-limit-var</td>
<td>OMP_THREAD_LIMIT</td>
<td>Implementation defined.</td>
<td>target and teams constructs</td>
<td>omp_get_thread_limit()</td>
<td>Data env.</td>
<td>![21.1.8] [6.10]</td>
</tr>
<tr>
<td>thread-num-var</td>
<td>(none)</td>
<td>zero</td>
<td>(none)</td>
<td>omp_get_thread_num()</td>
<td>Impl. Task</td>
<td>--</td>
</tr>
<tr>
<td>tool-libraries-var</td>
<td>OMP_TOOL_LIBRARIES</td>
<td>empty string</td>
<td>(none)</td>
<td>(none)</td>
<td>Global</td>
<td>![21.3.2] [6.10]</td>
</tr>
<tr>
<td>tool-var</td>
<td>OMP_TOOL</td>
<td>enabled</td>
<td>(none)</td>
<td>(none)</td>
<td>Global</td>
<td>![21.3.1] [6.18]</td>
</tr>
<tr>
<td>tool-verbose-init-var</td>
<td>OMP_TOOL_VERBOSE_INIT</td>
<td>disabled</td>
<td>(none)</td>
<td>(none)</td>
<td>Global</td>
<td>![21.3.3] [6.20]</td>
</tr>
<tr>
<td>wait-policy-var</td>
<td>OMP_WAIT_POLICY</td>
<td>Implementation defined.</td>
<td>(none)</td>
<td>(none)</td>
<td>Device</td>
<td>![21.2.3] [6.7]</td>
</tr>
</tbody>
</table>
Using OpenMP Tools

A tool indicates its interest in using the OMPT interface by providing a non-null pointer to an `ompt_start_tool_result` structure to an OpenMP implementation as a return value from the `ompt_start_tool` function.

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 (via `omp_tool_libraries`).

You can use `omp_tool_verbose_init` to help understand issues with loading or activating tools. This runtime library routine sets the tool-verbose-init-var ICV, which controls whether an OpenMP implementation will verbosely log the registration of a tool.

OpenMP API 5.2 Reference Guide Index

Clauses-9

Directives and Constructs-1–8

- allocate, allocators-1
- assume, assumes-2
- atomic-6
- barrier-5
- begin assumes-2
- begin declare target-2
- cancel-6
- Cancellation constructs
  - cancel-6
  - cancellation point-6
- cancellation point-6
- Combined forms-6–8
- critical-5
- Data environment directives-1
  - declare mapper-1
  - declare reduction-1
  - scan-1
  - threadprivate-1
- declare mapper-1
- declare reduction-1
- declare simd-2
- declare target-2
- declare variant-2
- depobj-5
- Device directives and construct-4
  - target-5
  - target data-4
  - target enter data-5
  - target exit data-5
  - target update-5
- dispatch-2
- distribute-4
- do-4
- flush-5
- for-4
- Informational & utility directives-2
  - assume, [begin] assumes-2
  - error-3
  - nothing-3
  - requires-2
- interop-5

Loop transformation constructs-3
- tile-3
- unroll-3
- masked-3

Memory management directives-1
- allocate-1
- allocators-1
- Memory spaces-1
- metadirective-1
- nothing-3
- ordered-6
- parallel-3
- Parallelism constructs-3
  - masked-3
  - parallel-3
  - simd-3
  - teams-3
  - requires-2
  - scan-1
  - scope-3
  - section, sections-4
  - simd-3
  - single-3

Synchronization constructs-5
- atomic-6
- barrier-5
- critical-5
- depobj-5
- flush-5
- ordered-6
- taskgroup-5
- taskwait-5
- target-5
- target data-4
- target enter data-5
- target exit data-5
- target update-5
- task-4
- taskgroup-5
- Tasking constructs-4
  - task-4
  - taskloop-4
  - taskyield-4
  - teams-3
  - threadprivate-1
  - tile-3
  - unroll-3

Variant directives-1
- [begin] declare target-2
- declare simd-2
- declare variant-2
- dispatch-2
- metadirective-1
- Work-distribution constructs-3
  - distribute-4
  - do-4
  - for-4
  - loop-4
  - scope-3
  - section, sections-4
  - single-3
  - workshare-3
  - workshare-3

Environment Variables-14–15

Internal Control Variable (ICV) Values-15

Runtime Library Routines-10–14

- Device information routines-11
- Device memory routines-11–12
- Environment display routine-14
- Event routine-13
- Interoperability routines-13
- Lock routines-12–13
- Memory management routines-13
- Resource relinquishing routines-11
- Tasking routines-11
- Teams region routines-11
- Thread affinity routines-10–11
- Thread team routines-10
- Timing routines-13
- Tool control routine-14