An OpenMP executable directive applies to the succeeding structured block. A structured-block is an OpenMP construct or a block of executable statements with a single entry at the top and a single exit at the bottom. OpenMP directives except simd and any declarative directive may not appear in Fortran.

### Variant directives

**metadirective** [2.3.4] [2.3.4]

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

- **#pragma omp metadirective**
  - clause (context-selector-specification) ...
  - or -
  - **#pragma omp begin metadirective** (context-selector-specification) ...
  - **#pragma omp end metadirective**

**declare variant** [2.3.5] [2.3.5]

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

- **#pragma omp declare variant** (variant-proc-name) ...
  - clause (context-selector-specification) ...

**派遣 [2.3.6]**

Controls whether variant substitution occurs for a given call.

- **#pragma omp dispatch** (clause) ...
  - stmt(s)

### Informational and utility directives

**requires** [2.5.1] [2.4]

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

- **#pragma omp requires** (clause) ...

**assumes and assume** [2.5.2]

Provides invariants to the implementation that may be used for optimization purposes.

- **#pragma omp assumes** (clause) ...
  - or -
  - **#pragma omp begin assumes** (clause) ...
  - **#pragma omp assumes** (clause) ...
  - or -

**nothing** [2.5.3]

Indicates explicitly that the intent is to have no effect.

- **#pragma omp nothing**

### Error directives

**error** [2.5.4]

Instructs the compiler or runtime to display a message and to perform an error action.

- **#pragma omp error** (clause) ...

**parallel construct** [2.6] [2.6]

Creates a team of OpenMP threads that execute the region.

- **#pragma omp parallel** (clause) ...
  - structed-block

**teams construct** [2.7] [2.7]

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

- **#pragma omp teams** (clause) ...
  - structed-block

---

© 2020 OpenMP Arb

Openmp.org

Openmp API 5.1 Page 1

www.openmp.org
**Directives and Constructs (continued)**

**masked construct**

**masked** [2.8]  [2.16]

Specifies a structured block that is executed by a subset of the threads of the current team. In 5.0, this is the master construct, in which master replaces masked.

```c
#pragma omp masked [filter(integer-expression)] structured-block
```

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.9.3</td>
<td>workshare: Divides the execution of the enclosed structured block into separate units of work, each executed only once by one thread.</td>
</tr>
<tr>
<td>2.8.3</td>
<td>workshare: loosely structured-block</td>
</tr>
<tr>
<td>2.8.4</td>
<td>workshare: nowait</td>
</tr>
<tr>
<td>2.9.4</td>
<td>workshare: strictly structured-block</td>
</tr>
<tr>
<td>2.9.5</td>
<td>workshare: nowait</td>
</tr>
</tbody>
</table>

**scope construct**

**scope** [2.9]

Defines a structured block that is executed by all threads in a team where additional OpenMP operations can be specified.

```c
#pragma omp scope [clause...] structured-block
```

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.8.3</td>
<td>scope: any of the clauses accepted by distribute or simd.</td>
</tr>
<tr>
<td>2.8.3</td>
<td>scope: any of the clauses accepted by distribute or simd.</td>
</tr>
</tbody>
</table>

**Worksharing constructs**

**sections** [2.10.1]  [2.1.1]

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.

```c
#pragma omp sections [clause...] { ... }
```

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.10.3</td>
<td>sections: workshare: Divides the execution of the enclosed structured block into separate units of work, each executed only once by one thread.</td>
</tr>
<tr>
<td>2.8.3</td>
<td>workshare: loosely structured-block</td>
</tr>
<tr>
<td>2.8.4</td>
<td>workshare: nowait</td>
</tr>
<tr>
<td>2.9.4</td>
<td>workshare: strictly structured-block</td>
</tr>
<tr>
<td>2.9.5</td>
<td>workshare: nowait</td>
</tr>
</tbody>
</table>

**for and do** [2.11.4]  [2.9.2]

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

```c
#pragma omp for [clause...] loop-nest
```

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.11.5</td>
<td>for simd and do simd: Specifies that the iterations of associated loops will be executed in parallel by threads in the team and the iterations executed by each thread can be executed concurrently using SIMD instructions.</td>
</tr>
<tr>
<td>2.11.5</td>
<td>for simd and do simd: Specifies that the iterations of associated loops will be executed in parallel by threads in the team and the iterations executed by each thread can be executed concurrently using SIMD instructions.</td>
</tr>
</tbody>
</table>

**declare simd** [2.11.5.3]  [2.9.3.3]

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 in a SIMD loop.

```c
#pragma omp declare simd [clause...] function declaration or declaration
```

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.11.5</td>
<td>declare simd: Any of the clauses accepted by the simd, for, or do directives.</td>
</tr>
<tr>
<td>2.11.5</td>
<td>declare simd: Any of the clauses accepted by the simd, for, or do directives.</td>
</tr>
</tbody>
</table>

**distribute loop constructs**

**distribute** [2.11.6.4]  [2.9.4.1]

Specifies loops which are executed by the initial teams.

```c
#pragma omp distribute [clause...] loop-nest
```

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.11.5</td>
<td>distribute loop constructs: SIMD directives and constructs</td>
</tr>
</tbody>
</table>

**SIMD directives and constructs**

**simd** [2.11.5.1]  [2.9.3.1]

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

```c
#pragma omp simd [clause...] loop-nest
```

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.11.5</td>
<td>simd: Any of the clauses accepted by distribute or simd.</td>
</tr>
<tr>
<td>2.11.5</td>
<td>simd: Any of the clauses accepted by distribute or simd.</td>
</tr>
</tbody>
</table>

**distribute parallel for** and **distribute parallel do**

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...] loop-nest
```

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.11.6</td>
<td>distribute parallel for: And distribute parallel do: These constructs specify a loop that can be executed in parallel by multiple threads that are members of multiple teams.</td>
</tr>
<tr>
<td>2.11.6</td>
<td>distribute parallel for: And distribute parallel do: These constructs specify a loop that can be executed in parallel by multiple threads that are members of multiple teams.</td>
</tr>
</tbody>
</table>
Directives and Constructs (continued)

Tasking constructs

task [2.12.1] [2.10.1]
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.

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

\$omp task [clause [,clause] ... ]
structured-block

```
#pragma omp task end
```

\$omp task end

\$omp task [clause [,clause] ... ]
- or -

\$omp task [clause [,clause] ... ]
strictly-structured-block

\$omp task end

clause: Any accepted by the distribute, parallel for simd, or parallel do simd directives.

loop construct

loop [2.11.7] [2.9.5]
Specifies that the iterations of the associated loops may
execute concurrently and permits the encountering
of execution of a different task.

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

\$omp loop [clause [,clause] ... ]
loop-nest

```
#pragma omp end loop
```

\$omp end loop

clause: Any accepted by the distribute, parallel for simd, or parallel do simd directives.

scan directive

scan [2.11.8] [2.9.6]
Specifies that scan computations update the list items
on each iteration of an enclosing loop nest associated with
a worksharing-loop, worksharing-loop SIMD, or simd directive.

```c
{ [structured-block-sequence
#pragma omp scan clause
structured-block-sequence ]
```

\$omp scan clause
structured-block-sequence

```
#pragma omp end scan
```

\$omp end scan

clause: inclusive (list) or exclusive (list)

Loop transformation constructs

tile [2.11.9.1]
Tiles one or more loops.

```c
#pragma omp tile sizes [size-list]
```

\$omp tile sizes [size-list]
loop-nest

```
#pragma omp end tile
```

\$omp end tile

clause: full partial [unroll-factor]

unroll [2.11.9.2]
Fully or partially unrolls a loop.

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

\$omp unroll [clause]
loop-nest

```
#pragma omp end unroll
```

\$omp end unroll

clause: Any accepted by the simd or taskloop directives.

taskyield [2.12.4] [2.10.4]
Specifies that the current task can be suspended in favor
of execution of a different task.

```c
#pragma omp taskyield
```

\$omp taskyield

clause: Any accepted by the distribute, parallel for simd, or parallel do simd directives.

Memory management directives

Memory spaces [2.13.1] [2.11.1]
Predefined memory spaces [Table 2.8, below] represent storage
resources for storage and retrieval of variables.

<table>
<thead>
<tr>
<th>Memory space</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>*omp_default_mem_space</td>
<td>Default storage</td>
</tr>
<tr>
<td>*omp_large_cap_mem_space</td>
<td>Large capacity</td>
</tr>
<tr>
<td>*omp_const_mem_space</td>
<td>Variables with constant values</td>
</tr>
<tr>
<td>*omp_high_bw_mem_space</td>
<td>High bandwidth</td>
</tr>
<tr>
<td>*omp_low_lat_mem_space</td>
<td>Low latency</td>
</tr>
</tbody>
</table>

allocate [2.13.3] [2.11.3]
Specifies how a set of variables is allocated.

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

\$omp allocate [clause [,clause] ... ]

```
#pragma omp end allocate
```

\$omp end allocate

clause: alloc (locator) or locals (locator-list)

Device directives and construct

target data [2.14.2] [2.12.2]
Creates a device environment for the extent of the region.

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

\$omp target data [clause [,clause] ... ]
structured-block

```
#pragma omp target end data
```

\$omp target end data

\$omp target data [clause [,clause] ... ]
loosely-structured-block

```
#pragma omp target end data
```

\$omp end target data

clause: map (locator-list) or -

```
#pragma omp target enter data
```

\$omp target enter data

clause: map (locator-list)

```
#pragma omp target parallel for simd
```

\$omp target parallel for simd

```
#pragma omp end target parallel for simd
```

\$omp end target parallel for simd

clause: Any accepted by the simd or taskloop directives.

allocate [2.13.3] [2.11.3]
Specifies how a set of variables is allocated.

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

\$omp allocate [clause [,clause] ... ]

```
#pragma omp end allocate
```

\$omp end allocate

clause: alloc (locator) or locals (locator-list)

Device directives and construct

target enter data [2.14.3] [2.12.3]
Maps variables to a device data environment.

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

\$omp target enter data [clause [,clause] ... ]
structured-block

```
#pragma omp target end data
```

\$omp target end data

clause: map (locator-list)

Directives and Constructs (continued)

Unmaps variables from a device data environment.

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

clause:
- map [ [map-type-modifier[, ...] ] map-type : locator-list]
- depend [ [depend-modifier[, ...] ] dependence-type : locator-list]
- nowait

C/C++

```c
|$omp target exit data [clause[,clause] ...]
```

Fortran

```fortran
!$omp target exit data ...
```

target [2.14.5] [2.12.5]
Maps variables to a device data environment and execute the construct on that device.

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

clause:
- private (list) firstprivate (list)
- in_reduction [reduction-identifier : list]
- map [ [map-type-modifier[, ...] ] map-type : locator-list]
- is_device_ptr (list) has_device_addr (list)
- defaultmap (implicit-behavior : variable-category)
- nowait
depend (depend-modifier[, ...] ) dependence-type : locator-list
allocate (locator : list) uses allocators (locator : allocators-tasks-array ) allocators (locator : allocators-tasks-array )

C/C++

```c
|$omp target [clause[,clause] ...] loosely-structured-block
```

Fortran

```fortran
!$omp target ...
```

|$omp target [clause[,clause] ...] strictly-structured-block

|$omp target [ clause[,clause] ...]

declare target [2.14.7] [2.12.7]
A declarative directive that specifies that variables, functions, and subroutines are mapped to a device.

```c
#pragma omp declare target declarations-definition-seq
#pragma omp declare target (extended-list)
```

```c
|$omp declare target (extended-list)
```

clause:
- to (extended-list) link (list) device_type (host | nohost | any)
- indirect (invoked-by-fptr)
- nowait
extended-list: A comma-separated list of named variables, procedure names, and named common blocks.
invoked-by-fptr: A constant boolean expression.
For A logical expression.

Interoperability construct

```
interop [2.15.1]
```

Retrieves interoperability properties from the OpenMP environment to enable interoperability with foreign execution contexts.

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

|$omp interop clause [[[],clause] ...]

clause:
- action-clause
device (integer-expression)
depend (depend-modifier[, ...] ) dependence-type : locator-list
init (interop-type) interop-type ([[], interop-type ... ] : interop-var)
destroy (interop-var)
use (interop-var)
nowait
interop-var: target targetsync
interop-modifier: prefer_type (preferrence-list)

Combined constructs

```
parallel for and parallel do [2.16.1] [2.13.1]
```
Specifies a parallel construct containing a worksharing-loop construct with a canonical loop nest and no other statements.

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

|$omp parallel do [clause[,clause] ...]

clause:
- any accepted by the parallel, for, or do directives except the nowait clause.

```
parallel loop [2.16.2] [2.13.2]
```
Shortcut for specifying a parallel loop construct containing a loop construct with a canonical loop nest and no other statements.

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

|$omp parallel loop [clause[,clause] ...]

clause:
- any accepted by the parallel or loop directives.
Directives and Constructs (continued)

**masked taskloop simd** [2.16.8]  
Shortcut for specifying a masked construct containing a taskloop simd construct and no other statements.

- `#pragma omp masked taskloop simd \\
  [clause], [clause] ... ] loop-nest`  
- `!$omp masked taskloop simd [clause], [clause] ... ] loop-nest`  
- `!$omp end masked taskloop simd`  

*clause:* Any clause used for the masked or taskloop simd directives.

**parallel masked taskloop** [2.16.9]  
Shortcut for specifying a parallel construct containing a masked taskloop construct and no other statements.

- `#pragma omp parallel masked taskloop \\
  [clause], [clause] ... ] loop-nest`  
- `!$omp parallel masked taskloop [clause], [clause] ... ] loop-nest`  
- `!$omp end parallel masked taskloop`  

*clause:* Any clause used for parallel or masked taskloop directives except the in_reduction clause.

**parallel masked taskloop simd** [2.16.18]  
Shortcut for specifying a parallel construct containing a masked taskloop simd construct and no other statements.

- `#pragma omp parallel masked taskloop simd \\
  [clause], [clause] ... ] loop-nest`  
- `!$omp parallel masked taskloop simd [clause], [clause] ... ] loop-nest`  
- `!$omp end parallel masked taskloop simd`  

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

**teams distribute** [2.16.11] [2.13.11]  
Shortcut for specifying a teams construct containing a distribute construct and no other statements.

- `#pragma omp teams distribute [clause], [clause] ... ] loop-nest`  
- `!$omp teams distribute [clause], [clause] ... ] loop-nest`  
- `!$omp end teams distribute`  

*clause:* Any clause used for the teams or distribute directives.

**teams distribute simd** [2.16.12] [2.13.12]  
Shortcut for specifying a teams construct containing a distribute simd construct and no other statements.

- `#pragma omp teams distribute simd \\
  [clause], [clause] ... ] loop-nest`  
- `!$omp teams distribute simd [clause], [clause] ... ] loop-nest`  
- `!$omp end teams distribute simd`  

*clause:* Any accepted by the teams or distribute simd directives.

**teams distribute parallel for** and **teams distribute parallel do** [2.16.13] [2.13.13]  
Shortcut for specifying a teams construct containing a distribute parallel worksharing-loop construct and no other statements.

- `#pragma omp teams distribute parallel for \\
  [clause], [clause] ... ] loop-nest`  
- `!$omp teams distribute parallel for [clause], [clause] ... ] loop-nest`  
- `!$omp end teams distribute parallel for`  

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

**teams distribute parallel for simd** and **teams distribute parallel do simd** [2.16.14] [2.13.14]  
Shortcut for specifying a teams construct containing a distribute parallel for simd or distribute parallel do simd construct and no other statements.

- `#pragma omp teams distribute parallel for simd \\
  [clause], [clause] ... ] loop-nest`  
- `!$omp teams distribute parallel for simd [clause], [clause] ... ] loop-nest`  
- `!$omp end teams distribute parallel for simd`  

*clause:* Any accepted by teams, distribute parallel for simd, or distribute parallel do simd directives, except for copypin.

**teams loop** [2.16.15] [2.13.15]  
Shortcut for specifying a teams construct containing a loop construct and no other statements.

- `#pragma omp teams loop [clause], [clause] ... ] loop-nest`  
- `!$omp teams loop [clause], [clause] ... ] loop-nest`  
- `!$omp end teams loop`  

*clause:* Any accepted by the teams or loop directives.

**teams loop simd** [2.16.15] [2.13.15]  
Shortcut for specifying a teams construct containing a loop simd construct and no other statements.

- `#pragma omp teams loop simd [clause], [clause] ... ] loop-nest`  
- `!$omp teams loop simd [clause], [clause] ... ] loop-nest`  
- `!$omp end teams loop simd`  

*clause:* Any accepted by the teams or loop directives.

**teams loop parallel** [2.16.16] [2.13.16]  
Shortcut for specifying a teams construct containing a parallel construct and no other statements.

- `#pragma omp teams loop parallel [clause], [clause] ... ] structured-block`  
- `!$omp teams loop parallel [clause], [clause] ... ] structured-block`  
- `!$omp end teams loop parallel`  

*clause:* Any accepted by the teams or loop directives.

**teams loop parallel for** and **teams loop parallel do** [2.16.17] [2.13.17]  
Shortcut for specifying a teams construct containing a parallel worksharing-loop construct and no other statements.

- `#pragma omp teams loop parallel for [clause], [clause] ... ] loop-nest`  
- `!$omp teams loop parallel for [clause], [clause] ... ] loop-nest`  
- `!$omp end teams loop parallel for`  

*clause:* Any accepted by the target, parallel for, or parallel do directives, except for copypin.

**target parallel loop** [2.16.18] [2.13.19]  
Shortcut for specifying a target construct containing a parallel loop construct and no other statements.

- `#pragma omp target parallel loop [clause], [clause] ... ] loop-nest`  
- `!$omp target parallel loop [clause], [clause] ... ] loop-nest`  
- `!$omp end target parallel loop`  

*clause:* Any accepted by the target or parallel loop directives except for copypin.

**target simd** [2.16.20] [2.13.20]  
Shortcut for specifying a target construct containing a simd construct and no other statements.

- `#pragma omp target simd [clause], [clause] ... ] loop-nest`  
- `!$omp target simd [clause], [clause] ... ] loop-nest`  
- `!$omp end target simd`  

*clause:* Any accepted by the target or simd directives.

**target teams** [2.16.21] [2.13.21]  
Shortcut for specifying a target construct containing a teams construct and no other statements.

- `#pragma omp target teams [clause], [clause] ... ] structured-block`  
- `!$omp target teams [clause], [clause] ... ] structured-block`  
- `!$omp end target teams`  

*clause:* Any accepted by the target or teams directives.

**target teams distributed** [2.16.22] [2.13.22]  
Shortcut for specifying a target construct containing a teams distribute construct and no other statements.

- `#pragma omp target teams distribute [clause], [clause] ... ] loop-nest`  
- `!$omp target teams distribute [clause], [clause] ... ] loop-nest`  
- `!$omp end target teams distribute`  

*clause:* Any accepted by the target or teams distribute directives.

**target teams distributed simd** [2.16.23] [2.13.23]  
Shortcut for specifying a target construct containing a teams distribute simd construct and no other statements.

- `#pragma omp target teams distributed simd \\
  [clause], [clause] ... ] loop-nest`  
- `!$omp target teams distributed simd [clause], [clause] ... ] loop-nest`  
- `!$omp end target teams distributed simd`  

*clause:* Any accepted by the target or teams distribute simd directives.

**target teams loop** [2.16.24] [2.13.24]  
Shortcut for specifying a target construct containing a teams loop construct and no other statements.

- `#pragma omp target teams loop [clause], [clause] ... ] loop-nest`  
- `!$omp target teams loop [clause], [clause] ... ] loop-nest`  
- `!$omp end target teams loop`  

*clause:* Any clause used for target or teams loop directives.
target teams distribute parallel for and
target teams distribute parallel do [2.16.26] [2.13.25]
Shortcut for specifying a target construct containing
teams distribute parallel for, teams distribute parallel do and
no other statements.

```c
#pragma omp target teams distribute parallel for
clause:
```

```c
#pragma omp target teams distribute parallel do &
clause:
```

```c
#pragma omp end target teams distribute parallel do/
clause:
```

### atomics

**atomics** [2.19.7] [2.17.7]
Ensures a specific storage location is accessed atomically.

```c
#pragma omp atomic clause [...] statement
```

### flushing

**flush** [2.19.8] [2.17.8]
Makes a thread's temporary view of memory consistent with
memory, and enforces an order on the memory
operations of the variables.

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

### ordered

**ordered** [2.19.9] [2.17.9]
Specifies a structured block that is to be executed in loop
iteration order in a parallelized loop, or it specifies cross
iteration dependencies in a doacross loop nest.

```c
#pragma omp ordered clause [...] statement
```

### Cancellation constructs

**cancel** [2.20.1] [2.18.1]
Activates cancellation of the innermost enclosing region
of the type specified.

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

### Dependent objects

**depobj** [2.19.10.1] [2.17.10.1]
Stand-alone directive that initializes, updates, or destroys
an OpenMP depend object.

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

### Cancellation constructs (continued)

**C cancellation** [if-clause]
```c
#pragma omp cancel construct-type-clause [ if-clause]
```
Directives and Constructs (continued)

cancellation point [2.20.2] [2.18.2]
Introduces a user-defined cancellation point at which tasks check if cancellation of the innermost enclosing region of the type specified has been activated.

Data environment directives

threadprivate [2.21.2] [2.19.2]
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.

declare reduction [2.21.5.7] [2.19.5.7]
Declares a reduction-identifier that can be used in a reduction clause.

mapper [2.21.7.4] [2.19.7.3]
Declares a user-defined mapper for a given type, and may define a mapper-identifier for use in a map clause.

Notes
### Runtime Library Routines

#### Thread team routines

**omp_set_num_threads** [3.2.1] [3.2.2]
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 `threads-var` ICV of the current task to `num_threads`.

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

**omp_get_num_threads** [3.2.2] [3.2.2]
Returns the number of threads in the current 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** [3.2.3] [3.2.3]
Returns an upper bound on the number of threads that could be used to form a new team if a parallel construct without a `num_threads` clause were encountered after execution returns from this routine.

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

**omp_get_thread_num** [3.2.3] [3.2.3]
Returns the thread number of the calling thread, within the current team.

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

**omp_get_in_parallel** [3.2.5] [3.2.6]
Returns true if the active-levels-var ICV is greater than zero; otherwise it returns false.

```c
logical function omp_get_in_parallel ()
```

**omp_set_dynamic** [3.2.6] [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 `dynamic_threads` ICV.

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

**omp_get_dynamic** [3.2.7] [3.2.8]
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** [3.2.8] [3.2.9]
Returns true if cancellation is enabled; otherwise it returns false. ICV: `cancel-var`

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

### thread affinity routines

**omp_set_schedule** [3.2.11] [3.2.12]
Affects the schedule that is applied when runtime is used as schedule kind, by setting the value of the run-sched-var ICV.

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

**omp_get_schedule**

```c
int omp_get_schedule (void);
```

**omp_set_nested** [3.2.9] [3.2.10]
Enables or disables nested parallelism, by setting the `max-active-levels-var` ICV.

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

**omp_get_nested** [3.2.10] [3.2.11]
Returns whether nested parallelism is enabled or disabled. ICV: `max-active-levels-var`

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

**omp_get_team_size** [3.2.19] [3.2.20]
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** [3.2.20] [3.2.21]
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_set_active_level**

```c
int omp_set_active_level (int level);
```

**omp_get_max_active_levels** [3.2.21]
Returns the maximum number of nested active parallel regions on the device that enclose the task containing the call. ICV: `max-active-levels-var`

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

**omp_get_level** [3.2.17] [3.2.18]
Returns the number of nested parallel regions on the device that enclose the task containing the call. ICV: `levels-var`

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

**omp_get_ancestor_thread_num** [3.2.18] [3.2.19]
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_place_num_procs** [3.3.2] [3.2.25]
Returns the number of places available to the execution environment in the place list.

```c
int omp_get_place_num_procs (void);
```

**omp_get_thread_limit** [3.2.13] [3.2.14]
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_thread_limit (void);
```

**omp_get_proc_bind** [3.3.1] [3.2.23]
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_place_num_procs** [3.3.3] [3.2.25]
Returns the number of processors available to the execution environment in the specified place.

```c
int omp_set_place_num_procs (int place_num);
```

**omp_get_place_procs** [3.3.3] [3.2.25]
Returns the number of processors available to the execution environment in the specified place.

```c
int omp_get_place_procs (place_num);
```

**omp_get_place_procs** [3.3.4] [3.2.26]
Returns numerical identifiers of the processors available to the execution environment in the specified place.

```c
int omp_get_place_procs (int place_num, int *ids);
```
Runtime Library Routines (continued)

omp_get_place_num ([3.3.5] [3.2.27])
Returns the place number of the place to which the
encountering thread is bound.

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

omp_get_partition_num_places ([3.3.7] [3.2.29])
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 ([3.3.11] [3.2.33])
Returns the number of places in the place-partition-var ICV
of the current task.

```c
size_t omp_get_partition_num_places ( & place_nums);
```

omp_get_affinity_format ([3.3.8] [3.2.30])
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 ([3.3.9] [3.2.31])
Returns the value of the affinity-format-var ICV on the
device.

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

omp_display_affinity ([3.3.10] [3.2.32])
Prints the OpenMP thread affinity information using the
format specification provided.

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

omp_capture_affinity ([3.3.11] [3.2.33])
Prints the OpenMP thread affinity information into a buffer
using the format specification provided.

```c
size_t omp_capture_affinity (char *buffer, size_t size,
const char *format);
```

omp_set_affinity_format ([3.3.12] [3.2.34])
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_num_procs ([3.3.13] [3.2.35])
Returns the number of processors that are available to the
device; otherwise, it returns false.

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

omp_set_default_device ([3.3.14] [3.2.36])
Assigns the value of the default-device-var ICV, which
determines default target device.

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

omp_get_default_device ([3.3.15] [3.2.37])
Returns the value of the default-device-var ICV, which
determines the default target device.

```c
int omp_get_default_device (void);
```

omp_get_num_devices ([3.3.16] [3.2.38])
Returns the number of non-host devices available for
offloading code or data.

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

omp_get_device_num ([3.3.17] [3.2.39])
Returns the device number of the device on which the
calling thread is executing.

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

omp_is_initial_device ([3.3.18] [3.2.40])
Returns true if the current task is executing on the host
device; otherwise, it returns false.

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

omp_get_initial_device ([3.3.19] [3.2.41])
Returns a device number representing the host device.

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

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

omp_target_alloc ([3.3.20] [3.2.42])
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);
```

omp_get_device ([3.3.21] [3.2.43])
Returns the value associated with a device.

```c
int omp_get_device (void);
```

omp_get_num_threads ([3.3.22] [3.2.44])
Returns the number of OpenMP threads that can participate
in each contention group created by a teams construct.

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

omp_get_device调控 ([3.3.23] [3.2.45])
Returns the value of the device-contention-var ICV.

```c
int omp_get_device (void);
```

omp_get_num_teams ([3.3.24] [3.2.46])
Returns the number of teams in the place-partition-var ICV
of the current task.

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

omp_get_max_teams ([3.3.25] [3.2.47])
Returns the number of non-host devices available for
offloading code or data.

```c
int omp_get_max_teams (void);
```

omp_set_teams_thread_limit ([3.3.26] [3.2.48])
Sets the maximum number of OpenMP threads that can participate
in each contention group created by a teams construct.

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

omp_is_final ([3.3.27] [3.2.49])
Returns true if the current task is executing on the host
device; otherwise, it returns false.

```c
int omp_is_final (void);
```

omp_get_device ([3.3.28] [3.2.50])
Returns a device number representing the host device.

```c
int omp_get_device (void);
```

omp_get_device_num ([3.3.29] [3.2.51])
Returns the value associated with a device.

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

omp_get_num_devices ([3.3.30] [3.2.52])
Returns the number of non-host devices available for
offloading code or data.

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

omp_get_device ([3.3.31] [3.2.53])
Returns a device number representing the host device.

```c
int omp_get_device (void);
```

omp_get_num_threads ([3.3.32] [3.2.54])
Returns the number of OpenMP threads that can participate
in each contention group created by a teams construct.

```c
int omp_get_num_threads (int device_num);
```

omp_get_device ([3.3.33] [3.2.55])
Returns a device number representing the host device.

```c
int omp_get_device (void);
```

omp_get_num_devices ([3.3.34] [3.2.56])
Returns the number of non-host devices available for
offloading code or data.

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

omp_get_device ([3.3.35] [3.2.57])
Returns a device number representing the host device.

```c
int omp_get_device (void);
```

omp_get_num_threads ([3.3.36] [3.2.58])
Returns the number of OpenMP threads that can participate
in each contention group created by a teams construct.

```c
int omp_get_num_threads (int device_num);
```

omp_get_device ([3.3.37] [3.2.59])
Returns a device number representing the host device.

```c
int omp_get_device (void);
```

omp_get_num_devices ([3.3.38] [3.2.60])
Returns the number of non-host devices available for
offloading code or data.

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

omp_get_device ([3.3.39] [3.2.61])
Returns a device number representing the host device.

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

**omp_target_free** [3.8.2] [3.6.2]
Frees the device memory allocated by the `omp_target_alloc` routine.

```fortran
void *omp_target_free (void *device_ptr, int_device_num);
```

**omp_target_memcpy** [3.8.3] [3.6.3]
Copies a rectangular subvolume from a multi-dimensional device.

```fortran
void omp_target_memcpy (void *dst, const void *src, size_t t src_offset, size_t t src_dimensions,-size_t t dst_offset, size_t t dst_dimensions, const void *device_ptr, int_device_num, int depobj_count, int omp_depend_t (*depobj_list)[]);
```

**omp_target_memcpy_rect** [3.8.4] [3.6.4]
Copies memory between any combination of host and device pointers.

```fortran
int omp_target_memcpy_rect (void *dst, const void *src, size_t t src_length, size_t t src_device_num, int depobj_count, int omp_depend_t (*depobj_list)[]);
```

**omp_target_memcpy_rect_async** [3.8.5] [3.6.5]
Asynchronously performs a copy between any combination of host and device pointers.

```fortran
int omp_target_memcpy_rect_async (void *dst, const void *src, size_t t src_element_size, int num_dims, const void *device_ptr, int_device_num, int omp_depend_t (*depobj_list)[]);
```

**omp_target_memcpy_async** [3.8.6] [3.6.6]
Asynchronously performs a copy between any combination of host and device pointers.

```fortran
int omp_target_memcpy_async (void *dst, const void *src, size_t t src_offset, size_t t src_dimensions, size_t t dst_offset, size_t t dst_dimensions, const void *device_ptr, int_device_num, int depobj_count, int omp_depend_t (*depobj_list)[]);
```

**omp_target_allocation** [3.8.7] [3.6.7]
Performs a copy between any combination of host and device pointers asynchronously.

```fortran
int omp_target_memcpy_allocation (void *dst, const void *src, size_t t src_offset, size_t t src_dimensions, size_t t dst_offset, size_t t dst_dimensions, const void *device_ptr, int_device_num, int depobj_count, int omp_depend_t (*depobj_list)[]);
```

**omp_target_nest_lock** [3.8.8] [3.6.8]
Maps a device pointer, which may be returned from `omp_target_alloc` or implementation-defined runtime routines, to a host pointer.

```fortran
int omp_target_nest_lock (void *dst, const void *src, size_t t src_offset, size_t t src_device_num, int num_dims, int depobj_count, int omp_depend_t (*depobj_list)[]);
```

**omp_target_nest_lock_async** [3.8.9] [3.6.9]
Maps a device pointer, which may be returned from `omp_target_alloc` or implementation-defined runtime routines, to a host pointer.

```fortran
int omp_target_nest_lock_async (void *dst, const void *src, size_t t src_offset, size_t t src_device_num, int num_dims, int depobj_count, int omp_depend_t (*depobj_list)[]);
```

**omp_target_nest_lock_allocation** [3.8.10] [3.6.10]
Removes the association between a host pointer and a device address on a given device.

```fortran
int omp_target_nest_lock_deallocation (void *dst, const void *src, size_t t src_offset, size_t t src_device_num, int num_dims, int depobj_count, int omp_depend_t (*depobj_list)[]);
```

**omp_target_disconnect** [3.8.11] [3.6.11]
Returns the device pointer that is associated with a host pointer for a given device.

```fortran
int omp_target_disconnect (void *dst, const void *src, size_t t src_offset, size_t t src_device_num, int num_dims, int depobj_count, int omp_depend_t (*depobj_list)[]);
```

**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** [3.9.1] [3.3.1]

```fortran
void omp_init_lock (omp_lock_t *lock);
void omp_init_lock_with_hint (omp_lock_t *lock, omp_sync_hint_t hint);
```

**Initialize lock with hint** [3.9.2] [3.3.2]

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

**Destroy lock** [3.9.3] [3.3.3]

Ensure that the OpenMP lock is uninitialized.

```fortran
void omp_destroy_lock (omp_lock_t *lock);
void omp_destroy_lock_with_hint (omp_lock_t *lock, omp_sync_hint_t hint);
```

**Set lock** [3.9.4] [3.3.4]

Sets an OpenMP lock. The calling task region is suspended until the lock is set.

```fortran
void omp_set_lock (omp_lock_t *lock);
void omp_set_lock_with_hint (omp_lock_t *lock, omp_sync_hint_t hint);
```

**Unset lock** [3.9.5] [3.3.5]

Ensure that the OpenMP lock is initialized.

```fortran
void omp_unset_lock (omp_lock_t *lock);
void omp_unset_lock_with_hint (omp_lock_t *lock, omp_sync_hint_t hint);
```

**Copy lock** [3.9.6] [3.3.6]

Copies a lock.

```fortran
void omp_copy_lock (omp_lock_t *lock);
void omp_copy_lock_with_hint (omp_lock_t *lock, omp_sync_hint_t hint);
```
Runtime Library Routines (continued)

Test lock [3.9.6] [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_lock_init (omp_lock_t *lock);`

Timing routines

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

- `omp_get_wtime`[3.10.1] [3.4.1]
Returns elapsed wall clock time in seconds.

- `double omp_get_wtime ()`
- `double omp_get_wtime ()`
- `double omp_get_wtick ()`

Memory management routines

Memory management types

The `omp_alloctrait`_ type in C/C++ and `omp_alloctrait`_ type in Fortran define members named `key` and `val`, with these types and values:

C/C++

- `enum omp_alloctrait_key_t`
- `enum omp_alloctrait_val_t`

Fortran

- `type(omp_alloctrait_key_kind), integer(omp_allocator_handle_kind), value :: (key)`
- `type(omp_alloctrait_val_kind), integer(omp_allocator_handle_kind), value :: (value)`

Deallocates previously allocated memory.

- `void omp_free (void *ptr, void *allocator, void *owner)`
- `void *omp_alloc (size_t size, void *owner)`
- `void *omp_calloc (size_t nmemb, void *owner)`
- `void *ompAligned Alloc (size_t size, void *owner)`

Interoperability routines

- `int omp_get_num_interop_properties (omp_interop_t object)`

Retrieves the number of implementation-defined properties available for an `omp_interop_t` object.

- `int omp_get_interop_int (omp_interop_t object, int ret_code)`

Retrieves an integer property from an `omp_interop_t` object.

- `const char* omp_get_interop_str (omp_interop_t object, char* ret_code)`

Retrieves a string property from an `omp_interop_t` object.

- `const char* omp_get_interop_name (omp_interop_t object, char* ret_code)`

Retrieves a property name from an `omp_interop_t` object.

- `void *omp_get_default_allocator (void)`

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

- `void *omp_get_allocator (void)`

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

- `void *omp_set_default_allocator (void)`

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

- `void *omp_get_interop_type_desc (omp_interop_t object)`

Retrieves a description of the type of a property associated with an `omp_interop_t` object.

- `void *omp_alloc (size_t size, void *owner)`

Requests a memory allocation from a memory allocator.

- `void *omp_aligned_alloc (size_t size, void *owner)`

Requests a zero-initialized memory allocation from a memory allocator.

- `void *ompcalloc (size_t nmemb, size_t size, void *owner)`

Requests a zero-initialized memory allocation from an `omp_interop_t` object.
### Runtime Library Routines (continued)

#### omp_realloc [3.13.9]
Deallocates previously allocated memory and requests a memory allocation from a memory allocator.

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

#### omp_control_tool [3.14] [1.8]
Enables a program to pass commands to an active tool.

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

#### Environment display routine

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

### Clauses

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

#### Allocate clause [2.13.4] [2.11.4]
Allocate ([allocate list] list)
Specifies the memory allocator to be used to obtain storage for private variables of a directive.

allocate ([allocate-modifier [ , allocate-modifier-1] list])
Specifies the memory allocator to be used to obtain storage for private variables of a directive.

allocate-modifier:
- allocator (allocator)
  -去哪里 are expressions of:
  - integer(omp_allocator_handle_kind), value :: &
  - omp_allocator_handle_t
  - size_t
  - void

```Fortran```
where
```
allocator where
```

- ptr = omp_null_allocator;
```
```C```
where
```
allocator where
```

- ptr = omp_null_allocator;
```
```C/C++```
where
```
allocator where
```

- omp_allocator_handle_t
  - size_t
  - void

#### Data copying clauses [2.21.6] [2.19.6]

copin (list)
Copies the value of the primary thread’s threadprivate variable to the threadprivate variable of each other member of the team executing the parallel region.

```Fortran```
copin (list)
```C```
copin (list)
```C/C++```
copin (list)

#### Data sharing clauses [2.21.4] [2.19.4]

copyin (list)
Copies the value of the primary thread’s threadprivate variable to the threadprivate variable of each other member of the team executing the parallel region.

```Fortran```
copyin (list)
```C```
copyin (list)
```C/C++```
copyin (list)

#### Depend clause [2.19.11] [2.17.11]
Enforces additional constraints on the scheduling of tasks or loop iterations, establishing dependencies only between sibling tasks or between loop iterations.

```
declare (dependence-type)
```
declaration-type must be source.
```
derepend (dependence-type : vec)
```

dependence-type must be sink and vec is the iteration vector with form: x_1 [± d_1], x_2 [± d_2], . . . , x_n [± d_n]
```
[Depend Clause continued on next page]"
Clauses (continued)

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

- **inoutset**: If the storage location of at least one of the list items matches the storage location of a list item appearing in a `depend` clause with an `in`, `out`, `inout`, or `mutexinoutset` `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.

- **depoj**: The task dependences are derived from the `depend` clause specified in the `depoj` clause constructs that initialized dependences represented by the depend objects specified in the `depend` clause as if the `depoj` clauses of the `depoj` constructs were specified in the current construct.

**If clauses [2.18] [2.15]**

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.

**Order and Ordered clauses [2.11.3] [2.9.2]**

- **order**: `order (order-modifier| concurrent)`
  - `order-modifier`: reproducible, unconstrained
  - Specifies an expected order of execution for the iterations of the associated loops of a loop-associated directive.

- **ordered**: `[{}|]`
  - Indicates the loops or how many loops to associate with a construct.

**Reduction clauses [2.21.5] [2.19.5]**

- **in_reduction**: `(reduction-identifier : list)`
  - Specifies that a task participates in a reduction.
  - `reduction-identifier`: Same as for `reduction`

- **task_reduction**: `(reduction-identifier : list)`
  - Specifies a reduction among tasks.
  - `reduction-identifier`: Same as for `reduction`

- **reduction**: `(reduction-modifier | reduction-identifier : list)`
  - Specifies a reduction-identifier and one or more list items.
  - `reduction-modifier`: `inscan, 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: `+, *, .and., .or., .eql., .neq.,` or one of the following intrinsic procedure names: `max, min, land, lor, leq`.

**SIMD clauses [2.11.5] [2.9.3]**

Also see Data sharing clauses and If clauses in this guide.

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

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

- **inbranch**: Specifies that the function will always be called from inside a conditional statement of a SIMD loop. (Used in `declare simd`, not `simd`).

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

- **notinbranch**: Specifies that the function will never be called from inside a conditional statement of a SIMD loop. (Used in `declare simd`, not `simd`).

- **safelen**:
  - If used then no two iterations executed concurrently with SIMD instructions can have a greater distance in the logical iteration space than the value of `length`. (Not used in `declare simd`).

- **simdlen**:
  - A constant positive integer expression that specifies the preferred number of iterations to be executed concurrently.

- **uniform**: `(argument-list)`
  - Declares one or more arguments to have an invariant logical iteration space than the value of `length`. (Not used in `task`).

**Tasking clauses [2.12] [2.10]**

- **affinity**: `(allocator : locator-list)`
  - A hint to execute closely to the location of the list items. `allocator` is `iterators-definition`. (Not used in `taskloop`).

- **allocate**: `(locator-list)`
  - See `Allocate clause on page 12 of this guide.

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

- **default**: `(private | firstprivate | shared | none)`
  - See Data sharing clauses, page 12 of this guide.

- **depend**: `(depend-modifier | dependence-type : locator-list)`
  - `depend-modifier`: `inout ||, depend, in, out, dependobj`
  - See Dependences clause on page 12 of this guide. (Not used in `task`).

- **detach**: `(list)`
  - When the task is done it is still in the system, and so the other tasks waiting for it to be completed are not released. (Also see omp_fulfilled_event)

- **final**: `(scalar-logical-expression)`
  - For final `(scalar-logical-expression)`
  - The generated task will be a final task if the expression evaluates to true.

- **firstprivate**: `(list)`
  - See Data sharing clauses on page 12 of this guide.

**Iterators**

**Iterators [2.1.6] [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.

- **C++ iterator-type**: `A type name`.
  - `For iterator-type`: `A type specifier`.

**Grainsize**

Causes the number of logical loop iterations assigned to each created task to be greater than or equal to the minimum of the value of the `grain-size` expression and the number of logical loop iterations, but less than twice the value of the `grain-size` expression. `strict` forces use of exact grain size, except for last iteration. (Not used in `task`).

**C/C++**

If present, any thread in the team can resume the task region after a suspension.
### 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 (Tables 2.2, 2.2, and 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>Scope</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>Data env.</td>
<td>[6.3] [6.3]</td>
</tr>
<tr>
<td>• nest-var</td>
<td>OMP_NESTED</td>
<td>Implementation defined.</td>
<td>omp_set_nested()</td>
<td><em>omp_get_nested()</em></td>
<td>--</td>
<td>[6.9] [6.9]</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_max_threads()</td>
<td>Data env.</td>
<td>[6.2] [6.2]</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>Data env.</td>
<td>[6.1] [6.1]</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>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>[6.4] [6.4]</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>[6.6] [6.6]</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>[6.7] [6.7]</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>[6.18] [6.10]</td>
</tr>
<tr>
<td>max-active-levels-var</td>
<td>OMP_MAX_ACTIVE_LEVELS, OMP_NESTED, OMP_NUM_THREADS, OMP_PROC_BIND</td>
<td>Implementation defined.</td>
<td>omp_set_max_active_levels(), omp_set_nested()</td>
<td>omp_get_max_active_levels()</td>
<td>Device</td>
<td>Data env.</td>
</tr>
<tr>
<td>active-levels-var</td>
<td>(none)</td>
<td>zero</td>
<td>(none)</td>
<td>omp_get_active_level()</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>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>---</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>[6.11] [6.11]</td>
</tr>
<tr>
<td>display-affinity-var</td>
<td>OMP_DISPLAY_AFFINITY</td>
<td>false</td>
<td>(none)</td>
<td>[omp_get_affinity_format()]</td>
<td>Global</td>
<td>[6.12] [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>Global</td>
<td>[6.14] [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>Data env.</td>
<td>[6.15] [6.15]</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>[6.17] [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>Global</td>
<td>[6.18] [6.16]</td>
</tr>
<tr>
<td>tool-var</td>
<td>OMP_TOOL</td>
<td>enabled</td>
<td>(none)</td>
<td>(none)</td>
<td>Global</td>
<td>[6.18] [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>Global</td>
<td>[6.19] [6.19]</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>[6.20]</td>
</tr>
<tr>
<td>debug-var</td>
<td>OMP_DEBUG</td>
<td>disabled</td>
<td>(none)</td>
<td>(none)</td>
<td>Global</td>
<td>[6.21] [6.20]</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>thread-num-var</td>
<td>(none)</td>
<td>zero</td>
<td>(none)</td>
<td>omp_get_thread_num()</td>
<td>Device</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>implicit-task-var</td>
<td>(none)</td>
<td>true</td>
<td>(none)</td>
<td></td>
<td>Data env.</td>
<td>---</td>
</tr>
<tr>
<td>team-size-var</td>
<td>(none)</td>
<td>one</td>
<td>(none)</td>
<td>omp_get_num_threads()</td>
<td>Team</td>
<td>---</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>Impl. Task</td>
<td>[6.22] [6.21]</td>
</tr>
<tr>
<td>ntteams-var</td>
<td>OMP_NUM_TEAMS</td>
<td>zero</td>
<td>omp_set_num_teams()</td>
<td>omp_get_max_teams()</td>
<td>Device</td>
<td>[6.23]</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>[6.24]</td>
</tr>
</tbody>
</table>
Environment Variables

OMP_ALLOCATOR [6.22] [5.21]
OpenMP memory allocators can be used to make allocation requests. This environment variable sets the initial value of def-allocator-var ICV that specifies the default allocator for allocation calls, directives, and clauses that do not specify an allocator. The value is a predefined allocator or a predefined memory space optionally followed by one or more allocator traits.

- **Predefined memory spaces** are listed in Table 2.8
- **Allocator traits** are listed in Table 2.9
- **Predefined allocators** are listed in Table 2.10

**Examples**
```c
setenv OMP_ALLOCATOR omp_high_bw_mem_alloc
setenv OMP_ALLOCATOR \omp_large_cap_mem_space : alignment=16, \pinned=true
setenv OMP_ALLOCATOR \omp_high_bw_mem_space : pool_size=1048576, \fallback=omp_low_lat_mem_alloc
```

**Memory space names**

<table>
<thead>
<tr>
<th>Memory space name</th>
<th>Table 2.8</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>omp_default_mem_space</strong></td>
<td></td>
</tr>
<tr>
<td><strong>omp_large_cap_mem_space</strong></td>
<td></td>
</tr>
<tr>
<td><strong>omp_const_mem_space</strong></td>
<td></td>
</tr>
<tr>
<td><strong>omp_high_bw_mem_space</strong></td>
<td></td>
</tr>
<tr>
<td><strong>omp_low_lat_mem_space</strong></td>
<td></td>
</tr>
</tbody>
</table>

**Allocator traits & allowed values**

<table>
<thead>
<tr>
<th>Trait</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>sync_hint</strong></td>
<td>contended, uncontended, serialized, private</td>
</tr>
<tr>
<td><strong>access</strong></td>
<td>all, cgroupl, ptteam, thread</td>
</tr>
<tr>
<td><strong>pool_size</strong></td>
<td>Positive integer value (default is impl. defined)</td>
</tr>
<tr>
<td><strong>fallback</strong></td>
<td>default_fb: null_fb, abort_fb,_allocator_fb</td>
</tr>
<tr>
<td><strong>flb_data</strong></td>
<td>An allocator handle (No default)</td>
</tr>
<tr>
<td><strong>pin</strong></td>
<td>true, false</td>
</tr>
<tr>
<td><strong>partition</strong></td>
<td>environment, nearest, blocked, interleaved</td>
</tr>
</tbody>
</table>

**Predefined allocators w/ memory space and trait values**

<table>
<thead>
<tr>
<th>Predefined Allocators</th>
<th>Memory Space</th>
<th>Trait Values</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>omp_default_mem_alloc</strong></td>
<td><strong>omp_default_mem_space</strong></td>
<td>fallback=null_fb</td>
</tr>
<tr>
<td><strong>omp_large_cap_mem_alloc</strong></td>
<td><strong>omp_large_cap_mem_space</strong></td>
<td>(none)</td>
</tr>
<tr>
<td><strong>omp_const_mem_alloc</strong></td>
<td><strong>omp_const_mem_space</strong></td>
<td>(none)</td>
</tr>
<tr>
<td><strong>omp_high_bw_mem_alloc</strong></td>
<td><strong>omp_high_bw_mem_space</strong></td>
<td>(none)</td>
</tr>
<tr>
<td><strong>omp_low_lat_mem_alloc</strong></td>
<td><strong>omp_low_lat_mem_space</strong></td>
<td>(none)</td>
</tr>
<tr>
<td><strong>omp_cgroup_mem_alloc</strong></td>
<td>Implementation defined access=groupl</td>
<td></td>
</tr>
<tr>
<td><strong>omp_pteam_mem_alloc</strong></td>
<td>Implementation defined access=ptteam</td>
<td></td>
</tr>
<tr>
<td><strong>omp_thread_mem_alloc</strong></td>
<td>Implementation defined access=thread</td>
<td></td>
</tr>
</tbody>
</table>

**OMP_AFFINITY_FORMAT** [6.14] [5.14]
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: [%c][%s][%o][l][i][size/type], where the field type may be either the short or long names listed below: [Table 6.3] [6.2].

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

**OMP_DECL** [6.21] [5.20]
Sets the debug-var ICV. The value must 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] [5.15]
Sets the initial value of the default-device-var ICV that controls the default device number to use in device constructs.

**OMP_DISPLAY_AFFINITY** var [6.13] [5.13]
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** var [6.12] [5.12]
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 false, the runtime may also display vendor-specific variables. If var is false, no information is displayed.

**OMP_DYNAMIC** var [6.3] [5.3]
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** levels [6.8] [5.8]
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** level [6.16] [5.16]
Sets the initial value of the max-task-priority-var ICV that controls the use of task priorities.

- **OMP_NESTED** nested [6.9] [5.9]
  - Controls nested parallelism with max-active-levels var ICV.

**OMP_NUM_TEAMS** [6.23]
Sets the maximum number of teams created by a teams construct by setting the *teams-var ICV*.

**OMP_NUM_THREADS** list [6.2] [5.2]
Sets the initial value of the nthreads-var ICV for the number of threads to use for parallel regions.

**OMP_PLACES** places [6.5] [5.5]
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.

**OMP_PROC_BIND** policy [6.4] [5.4]
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. [For versions prior to 5.1, replace primary with master.]

**OMP_SCHEDULE** [modifier];kind, chunk] [6.1] [5.1]
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] [6.6] [5.6]
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 megabytes, and G is gigabytes. If unit is not specified, size is in units of K.

**OMP_TARGET_OFFLOAD** [6.17] [5.17]
Sets the initial value of the target-offload-var ICV. The value must be one of mandatory, disabled, or default.

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

**OMP_THREAD_LIMIT** [6.10] [5.10]
Sets the maximum number of OpenMP threads to use in a contention group by setting the **thread-limit-var ICV**.

**OMP_TOOL** [enabled] [disabled] [6.18] [5.18]
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** library-list [6.19] [5.19]
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_WAIT_POLICY** policy [6.7] [5.7]
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.
Tool Activation

Activating an OMPT Tool [4.2] [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 tool activation. The OMPT Interface also includes a monitoring interface for tracing activity on target devices (section 4.2.5).

Step 1. Determine whether to initialize [4.2.2] [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 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`).

Step 2. Initializing a first-party tool [4.2.3] [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] [4.2.4]

To monitor execution of an OpenMP program on the host device, a tool’s initializer must register to receive notification of events that occur as an OpenMP program executes. A tool can register callbacks for OpenMP events using the runtime entry point known as `ompt_set_callback`, which has the following possible return codes:

- `ompt_set_error`
- `ompt_set_never`
- `ompt_set_impossible`
- `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_t`. While this is a compromise, it is better than providing unique runtime entry points with a precise type signatures to set and get the callback for each unique runtime entry point type signature.