Directives and Constructs

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

Variant directives

- **metadirective**
  - **pragma omp metadirective**
    - A directive that can specify multiple directive variants, one of which may be conditionally selected to replace the metadirective based on the enclosing OpenMP context.
  - **#pragma omp metadirective**
    - Declares a specialized variant of a base function and the declarative directive may not appear in Fortran
  - **!$omp begin metadirective**
    - Begins multiple-programming construct
  - **!$omp metadirective**
    - Ends multiple-programming construct
  - **!$omp declare variant**
    - 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 declare variant**
    - Declares a specialized variant of a base function and the declarative directive may not appear in Fortran
  - **#pragma omp end declare variant**
    - Ends multiple-programming construct

- **dispatch**
  - **#pragma omp dispatch**
    - Controls whether variant substitution occurs for a given call.
  - **#pragma omp dispatch**
    - Declares a specialized variant of a base function and the declarative directive may not appear in Fortran
  - **!$omp begin dispatch**
    - Begins multiple-programming construct
  - **!$omp dispatch**
    - Ends multiple-programming construct

- **assume**
  - **#pragma omp assumes**
    - Provides invariants to the implementation that may be used for optimization purposes.
  - **#pragma omp assume**
    - Declares a specialized variant of a base function and the declarative directive may not appear in Fortran
  - **!$omp begin assumes**
    - Begins multiple-programming construct
  - **!$omp assumes**
    - Ends multiple-programming construct
  - **#pragma omp assume**
    - Declares a specialized variant of a base function and the declarative directive may not appear in Fortran
  - **#pragma omp end assumes**
    - Ends multiple-programming construct

- **error**
  - **#pragma omp error**
    - Instructs the compiler or runtime to display a message and to perform an error action.
  - **#pragma omp error**
    - Declares a specialized variant of a base function and the declarative directive may not appear in Fortran
  - **!$omp begin error**
    - Begins multiple-programming construct
  - **!$omp error**
    - Ends multiple-programming construct

- **parallel**
  - **#pragma omp parallel**
    - Creates a team of OpenMP threads that execute the region.
  - **#pragma omp parallel**
    - Declares a specialized variant of a base function and the declarative directive may not appear in Fortran
  - **!$omp begin parallel**
    - Begins multiple-programming construct
  - **!$omp parallel**
    - Ends multiple-programming construct
  - **#pragma omp parallel**
    - Declares a specialized variant of a base function and the declarative directive may not appear in Fortran
  - **#pragma omp end parallel**
    - Ends multiple-programming construct

- **teams**
  - **#pragma omp teams**
    - Creates a league of initial teams where the initial thread of each team executes the region.
  - **#pragma omp teams**
    - Declares a specialized variant of a base function and the declarative directive may not appear in Fortran
  - **!$omp begin teams**
    - Begins multiple-programming construct
  - **!$omp teams**
    - Ends multiple-programming construct
  - **#pragma omp teams**
    - Declares a specialized variant of a base function and the declarative directive may not appear in Fortran
  - **#pragma omp end teams**
    - Ends multiple-programming construct
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.]

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

scope construct

scope [2.9]
Defines a structured block that is executed by all threads in a team but where additional OpenMP operations can be specified.

```
#pragma omp scope [ clause[ ] ... ]
structed-block
```

Worksharing constructs

sections [2.10.1] [2.8.1]
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
structed-block-sequence
... }
```

single [2.10.2] [2.8.2]
Specifies that the associated structured block is executed by only one of the threads in the team.

```
#pragma omp single [ clause[ ] ... ]
structed-block
```

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.

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

for simd and do simd [2.11.5.2] [2.9.3.2]
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 also be executed concurrently using SIMD instructions.

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

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

```
#pragma omp simd end do [ nowait ]
```

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

distribute 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.

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

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

```
clause:
```
```
simdlen (length)
linear (linear-list): linear-step)
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
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] ... ]
```

- `structured-block`
- `loosely-structured-block`
- `structed-block`
- `structed-block`

#### taskloop [2.12.2] [2.10.2]

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

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

- `structured-block`
- `loosely-structured-block`
- `structed-block`
- `structed-block`

#### allocate [2.13.3] [2.11.3]

Specifies how a set of variables is allocated.

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

- `private`
- `firstprivate`
- `lastprivate`
- `ordered`
- `reduction`
- `in_reduction`
- `reduce`
- `in_reduction`
- `allocate`
- `allocatable`
- `align`

### 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>Storage Selection Intent</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>omp_default_mem_space</code></td>
<td>Default storage</td>
</tr>
<tr>
<td><code>omp_large_mem_space</code></td>
<td>Large capacity</td>
</tr>
<tr>
<td><code>omp_const_mem_space</code></td>
<td>Variables with constant values</td>
</tr>
<tr>
<td><code>omp_high_bw_mem_space</code></td>
<td>High bandwidth</td>
</tr>
<tr>
<td><code>omp_low_mem_space</code></td>
<td>Low latency</td>
</tr>
</tbody>
</table>

#### allocate [2.13.3] [2.11.3]

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

- `private`
- `firstprivate`
- `lastprivate`
- `ordered`
- `reduction`
- `in_reduction`
- `allocate`
- `allocatable`
- `align`

### Device directives and construct

#### target data [2.14.2] [2.12.2]

Creates a device data environment for the extent of the region.

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

- `structured-block`
- `loosely-structured-block`
- `structed-block`
- `structed-block`

#### target enter data [2.14.3] [2.12.3]

Maps variables to a device data environment.

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

- `structured-block`
- `loosely-structured-block`
- `structed-block`
- `structed-block`

### Loop transformation constructs

#### tile [2.11.9.1]

Tiles one or more loops.

```c
#pragma omp tile [size-list[ , clause] ... ]
```

- `structured-block-sequence`
- `structured-block-sequence`

#### unroll [2.11.9.2]

Fully or partially unrolls a loop.

```c
#pragma omp unroll [clause] loop-nest
```

- `structured-block`
- `structured-block`

#### 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
```

- `task`
- `yield`

#### tile [2.11.9.1]

Tiles one or more loops.

```c
#pragma omp tile [size-list[ , clause] ... ]
```

- `structured-block-sequence`
- `structured-block-sequence`

#### unroll [2.11.9.2]

Fully or partially unrolls a loop.

```c
#pragma omp unroll [clause] loop-nest
```

- `structured-block`
- `structured-block`

### 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] ... ]
```

- `structured-block`
- `loosely-structured-block`
- `structed-block`
- `structed-block`

### 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>Storage Selection Intent</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>omp_default_mem_space</code></td>
<td>Default storage</td>
</tr>
<tr>
<td><code>omp_large_mem_space</code></td>
<td>Large capacity</td>
</tr>
<tr>
<td><code>omp_const_mem_space</code></td>
<td>Variables with constant values</td>
</tr>
<tr>
<td><code>omp_high_bw_mem_space</code></td>
<td>High bandwidth</td>
</tr>
<tr>
<td><code>omp_low_mem_space</code></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 [list[ , clause] ... ]
```

- `private`
- `firstprivate`
- `lastprivate`
- `ordered`
- `reduction`
- `in_reduction`
- `allocate`
- `allocatable`
- `align`

### Device directives and construct

#### target data [2.14.2] [2.12.2]

Creates a device data environment for the extent of the region.

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

- `structured-block`
- `loosely-structured-block`
- `structed-block`
- `structed-block`

#### target enter data [2.14.3] [2.12.3]

Maps variables to a device data environment.

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

- `structured-block`
- `loosely-structured-block`
- `structed-block`
- `structed-block`
Directives and Constructs (continued)

**target exit data** [2.14.4] [2.12.4]
Unmaps variables from a device data environment.

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

clause:

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

**clause:**

For C/C++
- `clause`:
  - `parallel`
  - `loop`
  - `taskloop`

For Fortran
- `clause`:
  - `parallel`
  - `loop`
  - `taskloop`

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

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

clause:

- `structed-block`
- `loosely-structured-block`
- `strictly-structured-block`

**Target update** [2.14.6] [2.12.6]
Makes the corresponding list items in the device data environment consistent with their original list items, according to the specified motion clauses.

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

clause: motion-clause or one of:

- `nowait`
- `depend` [depend-modifier[, ... ]]
  - `depend-modifier`: `dependence-type : locator-list`
  - `dependence-type`:
    - `loop`
    - `task`

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

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

clause:

- `declaration-definition-seq`
  - `declaration-definition`
    - `identifier` :
      - `locator-list`
      - `locator-list`
  - `if` [invoked-by-fptr]
    - `dependency-type`
  - `strictly-structured-block`
  - `loosely-structured-block`

**parallel sections** [2.16.3] [2.13.3]
Shortcut for specifying a parallel construct containing a sections construct and no other statements.

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

clause:

- `structured-block`
  - `structured-block-sequence`
- `structured-block`
  - `structured-block-sequence`

**parallel workshare** [2.16.4] [2.13.4]
Shortcut for specifying a parallel construct containing a workshare construct and no other statements.

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

clause:

- `interop-type`
  - `loop`
  - `task`
  - `loop`
  - `task`

**parallel do** [2.16.5] [2.13.5]
Shortcut for specifying a parallel construct containing only one worksharing-loop SIMD construct.

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

clause:

- `loop`
  - `loop`

**parallel masked** [2.16.6]
Shortcut for specifying a parallel construct containing a masked construct and no other statements.

```
#pragma omp parallel masked [clause [,clause] ... ]
$omp parallel masked [clause [,clause] ... ]
```

clause:

- `structured-block`
- `loosely-structured-block`

**masked taskloop** [2.16.7]
Shortcut for specifying a taskloop construct containing a taskloop construct and no other statements.

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

clause:

- `loop`
  - `loop`

**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.

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

clause:

- `loop`
  - `loop`

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

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

clause:

- `loop`
  - `loop`

**parallel sections** [2.16.3] [2.13.3]
Shortcut for specifying a parallel construct containing a sections construct and no other statements.

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

clause:

- `structured-block`
  - `structured-block-sequence`
- `structured-block`
  - `structured-block-sequence`

**parallel workshare** [2.16.4] [2.13.4]
Shortcut for specifying a parallel construct containing a workshare construct and no other statements.

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

clause:

- `loop`
  - `loop`

**parallel do** [2.16.5] [2.13.5]
Shortcut for specifying a parallel construct containing only one worksharing-loop SIMD construct.

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

clause:

- `loop`
  - `loop`

**parallel masked** [2.16.6]
Shortcut for specifying a parallel construct containing a masked construct and no other statements.

```
#pragma omp parallel masked [clause [,clause] ... ]
$omp parallel masked [clause [,clause] ... ]
```

clause:

- `structured-block`
- `loosely-structured-block`

**masked taskloop** [2.16.7]
Shortcut for specifying a taskloop construct containing a taskloop construct and no other statements.

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

clause:

- `loop`
  - `loop`
**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
```

**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
```

**teams distribute** [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
```

**teams distribute parallel**  
```
#pragma omp teams distribute parallel do simd \\
[clause[,]clause] ... \\
loop-nest
```

**teams distribute parallel do**  
```
#pragma omp teams distribute parallel do simd \\
[clause[,]clause] ... \\
loop-nest
```

**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
```

**teams loop simd**  
```
#pragma omp teams loop simd \\
[clause[,]clause] ... \\
loop-nest
```

**teams loop parallel**  
```
#pragma omp teams loop parallel do \\
[clause[,]clause] ... \\
loop-nest
```

**teams loop parallel simd**  
```
#pragma omp teams loop parallel do simd \\
[clause[,]clause] ... \\
loop-nest
```

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

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

**teams parallel do**  
```
#pragma omp teams parallel do simd \\
[clause[,]clause] ... \\
loop-nest
```

**teams parallel do simd**  
```
#pragma omp teams parallel do simd \\
[clause[,]clause] ... \\
loop-nest
```

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

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

**target parallel do**  
```
#pragma omp target parallel do \\
[clause[,]clause] ... \\
structured-block
```

**target parallel do simd**  
```
#pragma omp target parallel do simd \\
[clause[,]clause] ... \\
structured-block
```

**target parallel loop**  
```
#pragma omp target parallel loop \\
[clause[,]clause] ... \\
loop-nest
```

**target parallel loop simd**  
```
#pragma omp target parallel loop simd \\
[clause[,]clause] ... \\
loop-nest
```

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

```
#pragma omp target simd \\
[clause[,]clause] ... \\
loop-nest
```

**target SIMD**  
```
#pragma omp target SIMD \\
[clause[,]clause] ... \\
loop-nest
```

**target SIMD and parallel SIMD**  
```
#pragma omp target SIMD and parallel SIMD \\
[clause[,]clause] ... \\
loop-nest
```

**target SIMD do**  
```
#pragma omp target SIMD do \\
[clause[,]clause] ... \\
loop-nest
```

**target SIMD loop**  
```
#pragma omp target SIMD loop \\
[clause[,]clause] ... \\
loop-nest
```

**target SIMD loop simd**  
```
#pragma omp target SIMD loop simd \\
[clause[,]clause] ... \\
loop-nest
```

**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
```

**target teams simd**  
```
#pragma omp target teams simd \\
[clause[,]clause] ... \\
structured-block
```

**target teams loop**  
```
#pragma omp target teams loop \\
[clause[,]clause] ... \\
loop-nest
```

**target teams loop simd**  
```
#pragma omp target teams loop simd \\
[clause[,]clause] ... \\
loop-nest
```

**target teams parallel**  
```
#pragma omp target teams parallel \\
[clause[,]clause] ... \\
structured-block
```

**target teams parallel simd**  
```
#pragma omp target teams parallel simd \\
[clause[,]clause] ... \\
structured-block
```

**target teams parallel loop**  
```
#pragma omp target teams parallel loop \\
[clause[,]clause] ... \\
loop-nest
```

**target teams parallel loop simd**  
```
#pragma omp target teams parallel loop simd \\
[clause[,]clause] ... \\
loop-nest
```

**target teams simd**  
```
#pragma omp target teams simd \\
[clause[,]clause] ... \\
loop-nest
```

**target teams simd loop**  
```
#pragma omp target teams simd loop \\
[clause[,]clause] ... \\
loop-nest
```

**target teams simd loop simd**  
```
#pragma omp target teams simd loop simd \\
[clause[,]clause] ... \\
loop-nest
```

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

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

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

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

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

```
#pragma omp teams parallel do simd \\
[clause[,]clause] ... \\
loop-nest
```

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

```
#pragma omp teams parallel do simd loop \\
[clause[,]clause] ... \\
loop-nest
```

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

```
#pragma omp teams parallel do simd loop simd \\
[clause[,]clause] ... \\
loop-nest
```

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

```
#pragma omp teams parallel do simd loop simd loop \\
[clause[,]clause] ... \\
loop-nest
```

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

```
#pragma omp teams parallel do simd loop simd loop simd \\
[clause[,]clause] ... \\
loop-nest
```

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

```
#pragma omp teams parallel do simd loop simd loop simd loop \\
[clause[,]clause] ... \\
loop-nest
```

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

```
#pragma omp teams parallel do simd loop simd loop simd loop simd \\
[clause[,]clause] ... \\
loop-nest
```
**Directives and Constructs (continued)**

**target teams distribute parallel for and target teams distribute parallel do**

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 | ] , [ clause ] ... ]  

#pragma omp target teams distribute parallel do &
| clause | ] , [ clause ] ... ]  

#pragma omp target end teams distribute parallel do/
```

**taskgroup**

Specifies a region which a task cannot leave until all its descendant tasks generated inside the dynamic scope of the region have completed.

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

**atomic**

Ensures a specific storage location is accessed atomically.

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

**critical**

Restricts execution of the associated structured block to a single thread at a time.

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

**memory-order-clause**

Activates cancellation of the innermost enclosing region.

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

**depoj**

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

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

**Cancellation constructs**

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

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

**Synchronization constructs**

This 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 ] [ ]  
```

**ordered**

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
#pragma ompordered [ clause [ , [ clause ] ... ] ]  
```

**C/C++**

stand-alone directive.

```c
#pragma omp depobj ( [ ] clause )  
```

**clauses**

Continued.
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.

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 `nthreads[ ]` ICV of the current task to `num_threads`.

- **void omp_set_num_threads (int num_threads);**
- **subroutine omp_set_num_threads (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.

- **int omp_get_num_threads (void);**
- **integer function omp_get_num_threads ();**

**omp_get_max_threads [3.2.3] [3.2.3]**

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

- **int omp_get_max_threads (void);**
- **integer function omp_get_max_threads ();**

**omp_get_thread_num [3.2.4] [3.2.4]**

Returns the thread number of the calling thread, within the current team.

- **int omp_get_thread_num (void);**
- **integer function omp_get_thread_num ();**

**omp_in_parallel [3.2.5] [3.2.6]**

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

- **int omp_in_parallel (void);**
- **logical function omp_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 dvar ICV.

- **void omp_set_dynamic (int dynamic_threads);**
- **subroutine omp_set_dynamic (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

- **int omp_get_dynamic (void);**
- **logical function omp_get_dynamic ();**

**omp_get_cancellation [3.2.8] [3.2.9]**

Returns whether cancellation is enabled; otherwise it returns false. ICV: cancel-var

- **int omp_get_cancellation (void);**
- **logical function omp_get_cancellation ();**

**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

- **int omp_get_level (void);**
- **integer function omp_get_level ();**

**omp_get_max_active_levels [3.2.14] [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.

- **void omp_set_max_active_levels (int max_levels);**
- **subroutine omp_set_max_active_levels (max_levels);**
- **integer function omp_get_max_active_levels ();**

**omp_get_team_size [3.2.19] [3.2.20]**

Returns, for a given nested level of the current thread, the number of processors available to the execution environment in the specified place.

- **int omp_get_team_size (int level);**
- **integer function omp_get_team_size (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

- **int omp_get_active_level (void);**
- **integer function omp_get_active_level ();**

**omp_get_place_num_procs [3.3.3] [3.2.25]**

Returns the thread affinity policy to be used for the subsequent nested parallel regions that do not specify a proc_bind clause.

- **int omp_proc_bind_t omp_get_proc_bind (void);**
- **integer (kind=omp_proc_bind_kind) & function omp_get_proc_bind ();**

**omp_get_num_places [3.3.2] [3.2.24]**

Returns the maximum number of places available to the execution environment in the place list.

- **int omp_get_num_places (void);**
- **integer function omp_get_num_places ();**

**omp_get_num_procs [3.3.3] [3.2.25]**

Returns the number of processors available to the execution environment in the specified place.

- **int omp_get_num_procs (int place_num);**
- **integer function omp_get_num_procs (place_num);**
- **integer place_num**

**omp_get_place_proc_ids [3.3.4] [3.2.26]**

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

- **void omp_get_place_proc_ids (int place_num);**
- **integer function omp_get_place_proc_ids();**
- **subroutine omp_get_place_proc_ids(place_num, ids);**
- **integer place_num**
- **integer ids (*)**

---

Continued
### Runtime Library Routines (continued)

**omp_get_place_num**

Returns the place number of the place to which the
encountering thread is bound.

```fortran
! Fortran
integer function omp_get_place_num ()
```

**omp_get_partition_num_places**

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**

Returns the list of place numbers corresponding to the
places in the place-partition-var ICV of the innermost implicit
region.

```c
void omp_get_partition_place_nums ( &
    integer place_nums[*] );
```

**omp_set_affinity_format**

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_display_affinity**

Prints the OpenMP thread affinity information into a buffer
format specification provided.

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

**omp_get_affinity_format**

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_get_information**

Returns the value of the information-var ICV.

```c
int omp_get_information ( void );
```

**omp_pause_resource**

Returns the maximum value that can be specified in the
priority clause.

```c
integer function omp_get_max_task_priority ()
```

**omp_get_max_task_priority**

Returns the maximum number of OpenMP threads available
for subsequent teams region routines
and the number of threads to be used for subsequent
tools that do not specify a num_teams clause.

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

**omp_set_teams_thread_limit**

Sets the maximum number of OpenMP threads that can
participate in each contention group created by a teams
construct by setting the value of teams-thread-limit-var ICV.

```c
int omp_set_teams_thread_limit (int thread_limit);
```

**omp_get_teams_thread_limit**

Returns an upper bound on the number of teams that could
be created by a teams construct without a num_teams
clause that is encountered after execution returns from this
routine. ICC-var teams-thread-limit-var.

```c
int omp_get_teams_thread_limit (void);
```

**omp_set_num_teams**

Assigns the value of the default-device-var ICV.

```c
int omp_set_num_teams ( void );
```

**omp_get_num_teams**

Returns the number of non-host devices available for
offloading code or data.

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

**omp_get_partition_num_places**

Returns the number of initial teams in the current
omp_get_num_teams

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

**omp_get_place_num**

Returns the place number of the place to which the
calling thread is bound.

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

**omp_get_information**

Returns the device number of the device on which the
calling thread is executing.

```c
int omp_get_information ( void );
```

**omp_is_initial_device**

Returns true if the current task is executing on the host
device; otherwise, it returns false.

```c
logical function omp_is_initial_device ()
```

**omp_get_device_num**

Returns a number representing the host device.

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

### Device information routines

**omp_get_num_procs**

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

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

**omp_set_default_device**

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**

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**

Returns the number of non-host devices available for
offloading code.

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

**omp_get_device_num**

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**

Returns true if the current task is executing on the host
device; otherwise, it returns false.

```c
logical function omp_is_initial_device ()
```

**omp_get_device_num**

Returns a number representing the host device.

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

### Resource relinquishing routines

**omp_set_resource**

Allocates memory in a device data environment and returns
pointers in the data environments of target devices.

```c
type(c_ptr) function omp_set_resource ( &
    c_size_t, c_int size, device_num );
```

**omp_target_alloc**

Allocates memory in a device data environment and returns
pointers in the data environments of target devices.

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

**omp_pause_resource**

Allows the runtime to relinquish resources used by
OpenMP on the specified device. Valid kind values include
omp_pause_soft and omp_pause_hard.

```c
int omp_pause_resource ( &
    device_num );
```

**omp_pause_resource**

Allows the runtime to relinquish resources used by
OpenMP on the specified device. Valid kind values include
omp_pause_soft and omp_pause_hard.

```c
int omp_pause_resource ( &
    device_num );
```

**omp_pause_resource**

Allows the runtime to relinquish resources used by
OpenMP on the specified device. Valid kind values include
omp_pause_soft and omp_pause_hard.

```c
int omp_pause_resource ( &
    device_num );
```
Runtime Library Routines (continued)

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

```c
void omp_target_free(void *device_ptr, int device_num);
```

**omp_target_is_present** [3.8.3] [3.6.3]
Tests whether a host pointer refers to storage that is mapped to a given device.

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

**omp_target_memcpy** [3.8.5] [3.6.4]
Copies memory between any combination of host and device pointers.

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

**omp_target_memcpy_rect** [3.8.6] [3.6.5]
Copies a rectangular subvolume from a multi-dimensional array to another multi-dimensional array.

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

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

```c
int omp_target_memcpy_async(void *dst, const void *src, size_t dst_offset, size_t src_offset, int dst_device_num, int src_device_num, int depobj_count, depobj_list_t depobj_list);
```

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

```c
int omp_target_memcpy_rect_async(void *dst, const void *src, size_t dst_offset, size_t src_offset, size_t dst_device_num, int src_device_num, int depobj_count, depobj_list_t depobj_list);
```

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

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

**omp_target_associate_ptr** [3.8.9] [3.6.6]
Maps a device pointer, which may be returned from **omp_target_alocate** or implementation-defined runtime routines, to a host pointer.

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

**omp_target_disassociate_ptr** [3.8.10] [3.6.7]
Removes the association between a host pointer and a device address on a given device.

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

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

```c
void *omp_get_mapped_ptr(const void *ptr, int device_num);
```

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

**Initialize lock** [3.9.1] [3.3.1]

```c
void omp_init_lock(omp_lock_t *lock);
void omp_init_lock(omp_lock_t *lock, omp_depend_t hint);
void omp_init_lock_with_hint(omp_lock_t *lock, omp_lock_kind hint);
```

**Destroy lock** [3.9.3] [3.3.3]
Ensure that the OpenMP lock is uninitialized.

```c
void omp_destroy_lock(omp_lock_t *lock);
void omp_destroy_lock(omp_lock_t *lock, omp_lock_kind hint);
```

**Set lock** [3.9.4] [3.3.4]
Sets an OpenMP lock. The calling task region is suspended until the lock is set.

```c
void omp_set_lock(omp_lock_t *lock);
void omp_set_lock(omp_lock_t *lock, omp_lock_kind hint);
```

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

```c
void omp_unset_lock(omp_lock_t *lock);
void omp_unset_lock(omp_lock_t *lock, omp_lock_kind hint);
```
Runtime Library Routines (continued)

Test lock [3.9.6] [3.3.6]
Attempts to set an OpenMP lock but do not suspend execution of the task executing the routine.

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

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.

Timing routines

- `omp_get_wtime (void);`
- `double omp_get_wtime ();`
- `double omp_get_wtick (void);`
- `double omp_get_wtime ()` [3.4.2]

Memory management routines

**Memory Management Types**

The `omp_alloca_t` struct is in C/C++ and `omp_alloca_t` type in Fortran to define members named key and value, with these types and values:

- `C/C++ enum omp_alloca_t_key_t`
  - `For integer omp_alloca_t_key_kind`
  - `omp_atk_x` where `x` may be one of `sync`, `hint`, `alignment`, `access`, `pool_size`, `fallback`, `fb_data`, `pinned`, `partition`

- `C/C++ enum omp_alloca_t_value_t`
  - `For integer omp_alloca_t_val_kind`
  - `omp_atv_x` where `x` may be one of `false`, `true`, `default`, `omp_atv_X`

**Event routines**

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

- `omp_fulfill_event` [3.11.1] [3.5.1]
  - `void _omp_fulfill_event (omp_event_handle_t *event);`
- `omp_event_t` [3.12.1]
  - `subroutine omp_event_fulfill (event)`

Interoperability routines

- `omp_get_num_interop_properties` [3.12.1.1]
  - `int omp_get_num_interop_properties (omp_interop_t interop);`
- `omp_get_interop_int` [3.12.2]
  - `int omp_get_interop_int (const omp_interop_t interop, omp_interop_property_t property_id, int *ret_code);`
- `omp_get_interop_ptr` [3.12.3]
  - `void *omp_get_interop_ptr (const omp_interop_t interop, omp_interop_property_t property_id, int *ret_code);`
- `omp_get_interop_str` [3.12.4]
  - `const char *omp_get_interop_str (const omp_interop_t interop, omp_interop_property_t property_id, int *ret_code);`
- `omp_get_interop_name` [3.12.5]
  - `const char *omp_get_interop_name (omp_interop_t interop, omp_interop_property_t property_id);`
- `omp_get_interop_type_desc` [3.12.6]
  - `const char *omp_get_interop_type_desc (omp_interop_t interop, omp_interop_property_t property_id);`
- `omp_get_interop_rc_desc` [3.12.7]
  - `const char *omp_get_interop_rc_desc (omp_interop_t interop, int ret_code);`

- `omp_alloc` and `omp_aligned_alloc` [3.13.6] [3.7.6]
  - Request a memory allocation from a memory allocator.

- `void *omp_alloc (size_t size, omp_allocator_handle_t *allocator);`
- `void *omp_aligned_alloc (size_t size, size_t alignment, omp_allocator_handle_t *allocator);`

- `omp_free` [3.13.7] [3.7.7]
  - Deallocates previously allocated memory.

- `void omp_free (void *ptr, omp_allocator_handle_t *allocator);`
- `void omp_free (void *ptr, omp_allocator_handle_t *allocator) = omp_null_allocator;`

- `omp_alloca` and `omp_aligned_alloca` [3.13.8]
  - Request a zero-initialized memory allocation from a memory allocator.

- `void *omp_alloca (size_t nmemb, size_t size, omp_allocator_handle_t *allocator);`
- `void *omp_aligned_alloca (size_t nmemb, size_t size, size_t alignment, omp_allocator_handle_t *allocator);`
- `void *omp_aligned_alloca (size_t nmemb, size_t size, size_t alignment, size_t size, omp_allocator_handle_t *allocator) = omp_null_allocator;`
Deallocates previously allocated memory and requests a memory allocation from a memory allocator.

```c
void *omp_realloc(void *ptr, size_t size, omp_allocator_handle_t alloc, void *free_alloc);
```

```c
void *omp_realloc(void *ptr, size_t size, omp_allocator_handle_t alloc = omp_null_allocator, omp_allocator_handle_t free_alloc = omp_null_allocator);
```

### Data sharing clauses

- **Allocate clause**
  - `allocate (allocate-modifier, ...)`
  - `allocate (allocate-modifier, ...)`
  - Specifies the memory allocator to be used to obtain storage for private variables of a directive.
  - `allocate (allocate-modifier, ...)`
  - `allocate (allocate-modifier, ...)`

- **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.

- **Copyprivate (list)**
  - Broadcasts a value from the data environment of one implicit task to the data environments of the other implicit tasks belonging to the parallel region.

- **Map (map-type-modifier, ...)**
  - Specifies how an original list item is mapped from the current task's data environment to a corresponding list item in the device data environment of the device identified by the construct.

- **Defaultmap (variable-category)**
  - Default determines the data-mapping attributes referenced in a task construct and would otherwise be implicitly determined.

- **Lastprivate (lastprivate-modifier, ...)**
  - Declares one or more list items to be private to an implicit task or SIMD lane, and causes the corresponding original list item to be updated after the end of the region.

- **Linear (linear-list)**
  - Linear specifies that the list item is assigned the value that the list item would have after sequential execution of the loop nest.

- **Depend clause**
  - Depend provides additional constraints on the scheduling of tasks or loop iterations, establishing dependences only between sibling tasks or between loop iterations.

- **Depend (dependence-type)**
  - Depend (dependence-type) must be source.

- **Depend (dependence-type : vec)**
  - Depend (dependence-type) must be sink and vec is the iteration vector with form: \( x_1 \{ \pm d_1 \}, x_2 \{ \pm d_2 \}, \ldots, x_n \{ \pm d_n \} \)

- **Depend (dependence-type : locator-list)**
  - Depend (dependence-type : locator-list) also depends on the iterator definition.

- **Depend (dependence-type : locator-list, depend-modifier)**
  - Depend (dependence-type : locator-list, depend-modifier)

- **Depend (dependence-type : locator-list, depend-modifier : iterator)**
  - Depend (dependence-type : locator-list, depend-modifier)

- **Depend (dependence-type : locator-list, depend-modifier : iterator, present, inout, mutexinitoutset, inoutset, depobj)**
  - Depend (dependence-type : locator-list, depend-modifier)

- **Depend (dependence-type : iterator, iterators-definition)**
  - Depend (dependence-type : iterator, iterators-definition)

[Depend Clause continued on next page]
Clusases (continued)

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

If 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.

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

Order and Ordered clauses [2.11.3] [2.9.2]
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.

order [ { } ]
Indicates the loops 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 and identifier or more list items.
reduction-modifier: inscan, task, default
C/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 iterator, one of the following operators: +, -, *, .and., .or., .eqv., .neqv., or one of the following intrinsic procedure names: max, min, land, lor, leor.

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 (n)
A constant positive integer expression that specifies how many loops are associated with the construct. (Not used in declare simd.)

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

lastprivate (list)
See Data sharing clauses on page 12 of this guide. (Not used in task.)

mergeable
Specifies that the generated task is a mergeable task.

nogroup
Prevents creation of implicit taskgroup region. (Not used in task.)

num_tasks (num-tasks)
Create as many tasks as the minimum of the num-tasks expression and the number of logical loop iterations. (Not used in task.)

priority (priority-value)
A hint to the runtime. Sets the maximum priority value.

private (list)
See Data sharing clauses on page 12 of this guide.

task (reduction: [reduction-modifier: ] reduction-identifier: list)
See Reduction Clauses on this page. (Not used in task.)

shared (list)
See Data sharing clauses, page 12 of this guide.

untied
If present, any thread in the team can resume the task region after a suspension.

Tasking clauses [2.12] [2.10]
affinity ([aff-modifier: ] locator-list)
A hint to execute closely to the location of the list items. aff-modifier is iterator (iterators-definition). (Not used in taskloop.)
allocate ([allocor: ] 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)
See Depend clause on page 12 of this guide. (Not used in taskloop.)
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)

c/C++ final (scalar-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 ]
iterator-specifier:
[ iterator-type: ] identifier = range-specification
range-specification: 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/C++ iterator-type: A type name.
For iterator-type: A type specifier.
### 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: \(<\text{ENV_VAR}>_-\text{DEV}[_{-<\text{device_num}>}]\). 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>• omp_get_nested()</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>(none)</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>[6.10] [6.10]</td>
</tr>
<tr>
<td>max-active-levels-var</td>
<td>OMP_MAX_ACTIVE_LEVELS, OMP_MAX_ACTIVE_LEVELS, OMP_PROC_BIND</td>
<td>Implementation defined.</td>
<td>omp_set_max_active_levels(), omp_get_max_active_levels()</td>
<td>Device</td>
<td>Data env.</td>
<td>[6.8] [6.8] [6.9] [6.9] [6.2] [6.2] [6.4] [6.4]</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() omp_get_partition_place_nums() omp_get_place_num_procs() omp_get_place_proc_ids()</td>
<td>Device</td>
<td>Data env.</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>[6.13] [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>[6.13] [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>Device</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>omp_get_max_task_priority()</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>[6.16] [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>[6.22] [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>(none)</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>nteams-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_TASKS_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>

#### Notes

---

© 2020 OpenMP ARB
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 `omp_allocatemem_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:algorithm=16,\pinned=true
setenv OMP_ALLOCATOR \omp_high_bw_mem_space:pool_size=1048576,\fallback=omp_low_lat_mem_alloc_fb,fb_data=omp_low_lat_mem_alloc
```

Memory space names [Table 2.8]
- omp_default_mem_space
- omp_large_cap_mem_space
- omp_const_mem_space

<table>
<thead>
<tr>
<th>Allocator traits &amp; allowed values (default shown in blue) [Table 2.9]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

Predefined allocators w/ memory space and trait values [Table 2.10]
- omp_default_mem_alloc
- omp_large_cap_mem_alloc
- omp_const_mem_alloc
- omp_high_bw_mem_alloc
- omp_low_lat_mem_alloc
- omp_cgroup_mem_alloc
- omp_pteam_mem_alloc
- omp_thread_mem_alloc

OMP_AFINTY_FORMAT format [6.14] [5.14]
Sets the initial value of the `omp_affinity_format` 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: [fF][iI][lL][sS][zZ][eE][tT][yY][pP][eE][xX][tT]. The field type may be either the short or long names listed below [Table 6.3] [6.2].

| t | n | thread_num |
| T | n | num_teams |
| L | a | ancestor_num |
| P | A | thread_affinity |
| H | i | native_thread_id |

OMP_CANCELLATION [6.11] [5.11]
Sets the initial value of the cancel-var ICV. The value must be `true` or `false`. If `true`, the effects of the `cancel` construct and of cancellation points are enabled and cancellation is activated.

OMP_DEBUG [6.21] [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 device [6.15] [5.15]
Sets the initial value of the `device`-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 specifier 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 must 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 `dynamic`-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_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_STACKSIZE size [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 kibytes, M is megabytes, and G is gigabytes. If `size` is not specified, `size` is in units of `K`.

OMP_THREADS_THREADS_LIMIT [6.24]
Sets the maximum number of OpenMP threads to use in each contention group created by a `threads` construct by setting the `teams-thread-limit-vars` ICV.

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

OMP_TOOL tool-verbose-init-var [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.

OMP_PROC_BIND policy [6.4] [5.4]
Sets the initial value of the `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 `scatter` in quotes. [For versions prior to 5.1, replace `primary` with `master`.]
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.