OpenMP API 5.2 Reference Guide

OpenMP is a scalable model that gives parallel programmers a simple and flexible interface for developing portable parallel applications in C/C++ and Fortran. OpenMP is suitable for a wide range of algorithms running on multicore nodes and chips, NUMA systems, GPUs, and other such devices attached to a CPU.

Getting Started

Navigating this reference guide

- Directives and Constructs
  - Clauses
  - Runtime Library Routines

OpenMP Examples Document

An Examples Document and a link to a GitHub repository with code samples is at link.openmp.org/examples51.

Directives and Constructs

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

Data environment directives

threadprivate [5.2] [2.21.2]

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

For C/C++

```c++
#pragma omp threadprivate ([list])
```

For Fortran

```fortran
$omp threadprivate ([list])
```

list:

- A comma-separated list of file-scope, namespace-scope, or static block-scope variables that do not have incomplete types.
- For A comma-separated list of named variables and named common blocks. Common block names must appear between slashes.

declare reduction [5.5.11] [2.21.5.7]

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

For C/C++

```c++
#pragma omp declare reduction ([list])
```

For Fortran

```fortran
$omp declare reduction ([list])
```

<table>
<thead>
<tr>
<th>list:</th>
</tr>
</thead>
<tbody>
<tr>
<td>A a comma-separated list of file-scope, namespace-scope, or static block-scope variables that do not have incomplete types.</td>
</tr>
<tr>
<td>For A comma-separated list of named variables and named common blocks. Common block names must appear between slashes.</td>
</tr>
</tbody>
</table>

allocate [6.6] [2.13.3]

Specifies how a set of variables is allocated.

For C/C++

```c++
#pragma omp allocate ([clause])
```

For Fortran

```fortran
$omp allocate ([clause])
```

<table>
<thead>
<tr>
<th>clause:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Align: An integer power of 2.</td>
</tr>
<tr>
<td>Allocators:</td>
</tr>
<tr>
<td>alloc (type alloc, alloc in a Fortran ALLOCATE statement)</td>
</tr>
<tr>
<td>allocate-stmt (allocate statement)</td>
</tr>
<tr>
<td>kind (kind alloc in a Fortran ALLOCATE statement)</td>
</tr>
</tbody>
</table>

allocateors [6.7]

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

For C/C++

```c++
#pragma omp allocators ([clause])
```

For Fortran

```fortran
$omp allocators ([clause])
```

<table>
<thead>
<tr>
<th>clause:</th>
</tr>
</thead>
<tbody>
<tr>
<td>allocate-stmt (allocate statement)</td>
</tr>
</tbody>
</table>

Variant directives

[begin ]metadirective [7.4.3, 7.4.4] [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.

For C/C++

```c++
#pragma omp metadirective ([clause])
```

For Fortran

```fortran
$omp metadirective ([clause])
```

<table>
<thead>
<tr>
<th>clause:</th>
</tr>
</thead>
<tbody>
<tr>
<td>when (context-selector-specification: [directive-variant])</td>
</tr>
</tbody>
</table>

...
Directives and Constructs (continued)

[begin | declare variant | [7.5.4-5] | [2.3.5]
Declares a specialized variant of a base function and the context in which it is used.

C/C++

#pragma omp declare variant (variant-func-id) \\
    [clause [ [ , clause ] ... ]]
#pragma omp declare variant (variant-func-id) \\
    [clause [ [ , clause ] ... ]]
#pragma omp begin declare variant clause-match \\
    declaration-definition-seq
#pragma omp end declare variant

clause:
adjust_args (adjust-op : argument-list)
    adjust-op: nothing, need_device_ptr
append_args (append-op, append-op ...
    append-op: interop
    interop-type [ , interop-type ...
enter
link
match (context-selector-specification)
    REQUIRED. Specifies how to adjust the arguments of the base function when a specified variant function is selected for replacement.

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

For variant-func-id
The name of a function variant that is a base language identifier.

clause-match:
match (context-selector-specification)
    REQUIRED match clause

C/C++

#pragma omp dispatch (clause [ [ , clause ] ... ]
    function-dispatch-structured-block
#pragma omp dispatch (clause [ [ , clause ] ... ]
    function-dispatch-structured-block

clause:
device_type (host | nohost | any)
    enter (extended-list)
        A comma-separated list of named variables, procedure names, and named common blocks.
        indirect(invoked-by-fptr)
            Determines if the procedures in an enter clause may be invoked indirectly.
        link (list)
            Supports compilation of functions called in a target region that refer to the list items.

    For the second C/C++ form of declare target, at least one clause must be enter or link.
    For begin declare target, the enter and link clauses are not permitted.

Informational and utility directives

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

C/C++

#pragma omp requires clause [ [ , clause ] ... ]

For

#pragma omp requires clause [ [ , clause ] ... ]

C

!$omp requires clause [ [ , clause ] ... ]

clause:
atomic_default_mem_order (seq_cst | acq_rel | relaxed)
dynamic_allocators

Enables memory allocators to be used in a target region without specifying the uses_allocators clause on the corresponding target construct. (See target on page 5 of this guide.)

reverse_offload

Requires an implementation to guarantee that if a target construct specifies a device clause in which the ancestor modifier appears, the target region can execute on the parent device of an enclosing target region. (See target on page 5.)

unified_address

Requires that all devices accessible through OpenMP API routines and directives use a unified address space.

unified_shared_memory

Guarantees that in addition to the requirement of unified_address, storage locations in memory are accessible to threads on all available devices.

assume, [begin |] assumes [8.3.2-4] [2.5.2]
Provides invariants to the implementation that may be used for optimization purposes.

C/C++

#pragma omp assumes clause [ [ , clause ] ... ]
    - or -
#pragma omp begin assumes clause [ [ , clause ] ... ]
    declaration-definition-seq
#pragma omp end assumes
    - or -
#pragma omp assume clause [ [ , clause ] ... ]
    structured-block

C

!$omp assumes clause [ [ , clause ] ... ]
    - or -
!$omp assume clause [ [ , clause ] ... ]
    loosely-structured-block
!$omp end assume
    - or -
!$omp assume clause [ [ , clause ] ... ]
    strictly-structured-block

[&end assume]

clause:
avoided (directive-name [ , directive-name ] ...)
    Lists directives absent in the scope.
contains (directive-name [ , directive-name ] ...)
    Lists directives likely to be in the scope.
holds (omp-logical-expression)
    An expression guaranteed to be true in the scope.
no_openmp
    Indicates that no OpenMP code is in the scope.
no_openmp_routines
    Indicates that no OpenMP runtime library calls are executed in the scope.
no_parallelism
    Indicates that no OpenMP tasks or SIMD constructs will be executed in the scope.
nothing [8.4] [2.5.3]
Indicates explicitly that the intent is to have no effect.

```fortran
!$omp nothing
```

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

```fortran
!$omp error [ clause [ [, | clause ] ... ] ]
```

clause:
- at (compile | execution)
- message (msg-string)
- severity (fatal | warning)

Loop transformation constructs

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

```fortran
!$omp tile clause
```

```fortran
!$omp end tile
```

clause:
- loop-nest

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

```fortran
!$omp unroll [ clause ]
```

```fortran
!$omp end unroll
```

clause:
- sizes (size-list)
- full
- partial (unroll-factor)

Parallelism constructs

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

```fortran
!$omp parallel [ clause [ [, | clause ] ... ] ]
```

```fortran
!$omp end parallel
```

clause:
- structured-block

```fortran
!$omp parallel [ clause [ [, | clause ] ... ] ]
```

```fortran
!$omp end parallel
```

clause:
- loosely-structured-block

```fortran
!$omp parallel [ clause [ [, | clause ] ... ] ]
```

```fortran
!$omp end parallel
```

clause:
- strictly-structured-block

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

```fortran
!$omp teams [ clause [ [, | clause ] ... ] ]
```

```fortran
!$omp end teams
```

clause:
- structured-block

```fortran
!$omp teams [ clause [ [, | clause ] ... ] ]
```

```fortran
!$omp end teams
```

clause:
- loosely-structured-block

```fortran
!$omp teams [ clause [ [, | clause ] ... ] ]
```

```fortran
!$omp end teams
```

clause:
- strictly-structured-block

```fortran
!$omp teams [ clause [ [, | clause ] ... ] ]
```

```fortran
!$omp end teams
```

clause:
- tile

simd [10.4] [2.11.5.1]
Applied to a loop to indicate that the loop can be transformed into a SIMD loop.

```fortran
!$omp simd [ clause [ [, | clause ] ... ] ]
```

```fortran
!$omp end simd
```

clause:
- loop-nest

Work-distribution constructs

single [11.1] [2.10.2]
Specifies that the associated structured block is executed by only one of the threads in the team.

```fortran
!$omp single [ clause [ [, | clause ] ... ] ]
```

```fortran
!$omp end single
```

clause:
- structured-block

```fortran
!$omp single [ clause [ [, | clause ] ... ] ]
```

```fortran
!$omp end single
```

clause:
- loosely-structured-block

```fortran
!$omp single [ clause [ [, | clause ] ... ] ]
```

```fortran
!$omp end single
```

clause:
- strictly-structured-block

```fortran
!$omp single [ clause [ [, | clause ] ... ] ]
```

```fortran
!$omp end single
```

clause:
- nothing

workshare [11.4] [2.10.3]
Divides the execution of the enclosed structured block into separate units of work, each executed only once by one thread.

```fortran
!$omp workshare [ clause [ [, | clause ] ... ] ]
```

```fortran
!$omp end workshare
```

clause:
- nothing

scope [11.3] [2.9]
Defines a structured block that is executed by all threads in a team but where additional OpenMP operations can be specified.

```fortran
!$omp scope [ clause [ [, | clause ] ... ] ]
```

```fortran
!$omp end scope
```

clause:
- nothing

masked [10.5] [2.8]
Specifies a structured block that is executed by a subset of the threads of the current team.

```fortran
!$omp masked [ clause ]
```

```fortran
!$omp end masked
```

clause:
- structured-block

```fortran
!$omp masked [ clause ]
```

```fortran
!$omp end masked
```

clause:
- loosely-structured-block

```fortran
!$omp masked [ clause ]
```

```fortran
!$omp end masked
```

clause:
- strictly-structured-block

```fortran
!$omp masked [ clause ]
```

```fortran
!$omp end masked
```

clause:
- filter (thread_num)

Choose which thread executes.
## Directives and Constructs (continued)

### section and sections [11.3] [2.10.1]
A non-iterative worksharing construct that contains a set of structured blocks that are to be distributed among and executed by the threads in a team.

### distribute [11.6] [11.1.6]
Sspecifies loop(s) which are executed by the initial teams.

```c
#pragma omp distribute [clause[…]]
```

#### Example

```c
#pragma omp distribute [private(list) collapse(n) lastprivate(list) firstprivate(list) reduction(list) schedule(list) nowait]
```

### do and for [11.5.1-2] [2.11.4]
Specifies that the iterations of associated loops will be executed in parallel by threads in the team.

```c
#pragma omp for [clause[…]]
```

#### Example

```c
#pragma omp for [private(list) collapse(n) lastprivate(list) firstprivate(list) schedule(list) nowait]
```

### loop [11.7] [2.11.1]
Sspecifies that the iterations of the associated loops may execute concurrently and permits the encountering thread(s) to execute the loop accordingly.

```
#pragma omp loop [clause[…]]
```

#### Example

```c
#pragma omp loop [private(list) collapse(n) lastprivate(list) firstprivate(list) schedule(list) nowait]
```

### Tasking constructs

#### task [12.5] [2.12.1]
Defines an explicit task. The data environment of the task is created according to the data-sharing attribute clauses on the task construct, per-data environment ICVs, and any defaults that apply.

```
#pragma omp task [clause[…]]
```

#### Example

```c
#pragma omp task [private(list) collapse(n) lastprivate(list) firstprivate(list) schedule(list) nowait]
```

### Device directives and construct
target data [13.5] [2.14.2]
Maps variables to a device data environment for the extent of the region.

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

#### Example

```c
#pragma omp target data [private(list) collapse(n) lastprivate(list) firstprivate(list) schedule(list) nowait]
```
target enter data [13.6] [2.14.3]
Maps variables to a device data environment.

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

[target update [13.9] [2.14.6]
Makes the corresponding list items in the device data environment consistent with their original list items, according to the specified motion clauses.

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

target exit data [13.7] [2.14.4]
Unmaps variables from a device data environment.

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

See Clause info on pg. 9

### Interoperability construct

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

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

See Clause info on pg. 9

### Synchronization constructs

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

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

barrier [15.3.1] [2.19.2]
Specifies an explicit barrier that prevents any thread in a team from continuing past the barrier until all threads in the team encounter the barrier.

```fortran
#pragma omp barrier
```

### Barrier construct

```fortran
!$omp critical
```

taskgroup [15.4] [2.19.6]
Specifies a region which a task cannot leave until all its descendant tasks generated inside the dynamic scope of the region have completed.

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

See Clause info on pg. 9

### Taskgroup construct

```fortran
!$omp taskgroup
```

### Taskwait construct

```fortran
!$omp taskwait
```

See Clause info on pg. 9

### Taskwait construct

```fortran
!$omp taskwait
```

See Clause info on pg. 9

### Taskwait construct

```fortran
!$omp taskwait
```

See Clause info on pg. 9

### Taskwait construct

```fortran
!$omp taskwait
```

See Clause info on pg. 9

### Taskwait construct

```fortran
!$omp taskwait
```

See Clause info on pg. 9

### Taskwait construct

```fortran
!$omp taskwait
```

See Clause info on pg. 9

### Taskwait construct

```fortran
!$omp taskwait
```

See Clause info on pg. 9

### Taskwait construct

```fortran
!$omp taskwait
```

See Clause info on pg. 9

### Taskwait construct

```fortran
!$omp taskwait
```

See Clause info on pg. 9

### Taskwait construct

```fortran
!$omp taskwait
```

See Clause info on pg. 9

### Taskwait construct

```fortran
!$omp taskwait
```

See Clause info on pg. 9

### Taskwait construct

```fortran
!$omp taskwait
```

See Clause info on pg. 9

### Taskwait construct

```fortran
!$omp taskwait
```

See Clause info on pg. 9

### Taskwait construct

```fortran
!$omp taskwait
```

See Clause info on pg. 9

### Taskwait construct

```fortran
!$omp taskwait
```

See Clause info on pg. 9

### Taskwait construct

```fortran
!$omp taskwait
```

See Clause info on pg. 9

### Taskwait construct

```fortran
!$omp taskwait
```

See Clause info on pg. 9

### Taskwait construct

```fortran
!$omp taskwait
```

See Clause info on pg. 9

### Taskwait construct

```fortran
!$omp taskwait
```

See Clause info on pg. 9

### Taskwait construct

```fortran
!$omp taskwait
```

See Clause info on pg. 9

### Taskwait construct

```fortran
!$omp taskwait
```

See Clause info on pg. 9

### Taskwait construct

```fortran
!$omp taskwait
```

See Clause info on pg. 9

### Taskwait construct

```fortran
!$omp taskwait
```

See Clause info on pg. 9

### Taskwait construct

```fortran
!$omp taskwait
```

See Clause info on pg. 9

### Taskwait construct

```fortran
!$omp taskwait
```

See Clause info on pg. 9

### Taskwait construct

```fortran
!$omp taskwait
```

See Clause info on pg. 9

### Taskwait construct

```fortran
!$omp taskwait
```

See Clause info on pg. 9

### Taskwait construct

```fortran
!$omp taskwait
```

See Clause info on pg. 9

### Taskwait construct

```fortran
!$omp taskwait
```

See Clause info on pg. 9

### Taskwait construct

```fortran
!$omp taskwait
```

See Clause info on pg. 9

### Taskwait construct

```fortran
!$omp taskwait
```

See Clause info on pg. 9

### Taskwait construct

```fortran
!$omp taskwait
```

See Clause info on pg. 9

### Taskwait construct

```fortran
!$omp taskwait
```

See Clause info on pg. 9

### Taskwait construct

```fortran
!$omp taskwait
```

See Clause info on pg. 9

### Taskwait construct

```fortran
!$omp taskwait
```

See Clause info on pg. 9

### Taskwait construct

```fortran
!$omp taskwait
```

See Clause info on pg. 9

### Taskwait construct

```fortran
!$omp taskwait
```

See Clause info on pg. 9

### Taskwait construct

```fortran
!$omp taskwait
```

See Clause info on pg. 9

### Taskwait construct

```fortran
!$omp taskwait
```

See Clause info on pg. 9

### Taskwait construct

```fortran
!$omp taskwait
```

See Clause info on pg. 9

### Taskwait construct

```fortran
!$omp taskwait
```

See Clause info on pg. 9

### Taskwait construct

```fortran
!$omp taskwait
```

See Clause info on pg. 9

### Taskwait construct

```fortran
!$omp taskwait
```

See Clause info on pg. 9

### Taskwait construct

```fortran
!$omp taskwait
```

See Clause info on pg. 9

### Taskwait construct

```fortran
!$omp taskwait
```

See Clause info on pg. 9

### Taskwait construct

```fortran
!$omp taskwait
```

See Clause info on pg. 9

### Taskwait construct

```fortran
!$omp taskwait
```

See Clause info on pg. 9

### Taskwait construct

```fortran
!$omp taskwait
```

See Clause info on pg. 9

### Taskwait construct

```fortran
!$omp taskwait
```

See Clause info on pg. 9

### Taskwait construct

```fortran
!$omp taskwait
```

See Clause info on pg. 9

### Taskwait construct

```fortran
!$omp taskwait
```

See Clause info on pg. 9

### Taskwait construct

```fortran
!$omp taskwait
```

See Clause info on pg. 9

### Taskwait construct

```fortran
!$omp taskwait
```

See Clause info on pg. 9

### Taskwait construct

```fortran
!$omp taskwait
```

See Clause info on pg. 9

### Taskwait construct

```fortran
!$omp taskwait
```

See Clause info on pg. 9

### Taskwait construct

```fortran
!$omp taskwait
```

See Clause info on pg. 9

### Taskwait construct

```fortran
!$omp taskwait
```

See Clause info on pg. 9

### Taskwait construct

```fortran
!$omp taskwait
```

See Clause info on pg. 9

### Taskwait construct

```fortran
!$omp taskwait
```

See Clause info on pg. 9

### Taskwait construct

```fortran
!$omp taskwait
```

See Clause info on pg. 9

### Taskwait construct

```fortran
!$omp taskwait
```

See Clause info on pg. 9

### Taskwait construct

```fortran
!$omp taskwait
```

See Clause info on pg. 9

### Taskwait construct

```fortran
!$omp taskwait
```

See Clause info on pg. 9

### Taskwait construct

```fortran
!$omp taskwait
```

See Clause info on pg. 9

### Taskwait construct

```fortran
!$omp taskwait
```

See Clause info on pg. 9

### Taskwait construct

```fortran
!$omp taskwait
```

See Clause info on pg. 9

### Taskwait construct

```fortran
!$omp taskwait
```

See Clause info on pg. 9

### Taskwait construct

```fortran
!$omp taskwait
```

See Clause info on pg. 9

### Taskwait construct

```fortran
!$omp taskwait
```

See Clause info on pg. 9

### Taskwait construct

```fortran
!$omp taskwait
```

See Clause info on pg. 9

### Taskwait construct

```fortran
!$omp taskwait
```

See Clause info on pg. 9

### Taskwait construct

```fortran
!$omp taskwait
```

See Clause info on pg. 9

### Taskwait construct

```fortran
!$omp taskwait
```

See Clause info on pg. 9

### Taskwait construct

```fortran
!$omp taskwait
```

See Clause info on pg. 9

### Taskwait construct

```fortran
!$omp taskwait
```

See Clause info on pg. 9

### Taskwait construct

```fortran
!$omp taskwait
```

See Clause info on pg. 9

### Taskwait construct

```fortran
!$omp taskwait
```

See Clause info on pg. 9

### Taskwait construct

```fortran
!$omp taskwait
```

See Clause info on pg. 9

### Taskwait construct

```fortran
!$omp taskwait
```

See Clause info on pg. 9

### Taskwait construct

```fortran
!$omp taskwait
```
atomic (15.8.4) [2.19.7]
Ensures a specific storage location is accessed atomically.

atomic clause: read, write, update
memory-order-clause: seq_cst, acc_rel, release, acquire, relaxed
extended-atomic: capture, compare, fail, weak

atomic (continued)
For capture-statement: Has the form \( v \leftarrow x \)
For statement:
if atomic clause is... statement:
read \( v \leftarrow x \)
write \( x \leftarrow v \)
update \( v \leftarrow \text{operator expression} \)
...!

!$omp end atomic

Cancellation constructs
cancel [16.1] [2.20.1]
Activates cancellation of the innermost enclosing region of the type specified.

!$omp cancel construct-type-clause[ , ] if-clause

!$omp end atomic

cancellation point [16.2] [2.20.2]
Introduces a user-defined cancellation point at which tasks check if cancellation of the innermost enclosing region of the type specified has been activated.

#$omp cancellation point construct-type-clause

Combined Constructs and Directives
The following combined constructs and directives are created following the parameters defined in section 17 of the OpenMP API version 5.2 specification and were explicitly defined in previous versions.

do simd and for simd [17] [2.11.5.2]
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.

#$omp do simd [clause[ , ] clause] ... loop-nest

#$omp for simd [clause[ , ] clause] ... loop-nest

distribution [17] [2.11.6.2]
Specifies a loop that will be distributed across the primary threads of the teams region and executed concurrently using SIMD instructions.

#$omp distribute simd [clause[ , ] clause] ... loop-nest

#$omp end distribute simd

distribution parallel do and distribute parallel for [17] [2.11.6.3]
Specify a loop that can be executed in parallel by multiple threads that are members of multiple teams.

#$omp distribute parallel do [clause[ , ] clause] ... loop-nest

#$omp end distribute parallel do

$omp #pragma omp atomic [clause[ , ] clause]...
statement

$omp atomic [clause[ , ] clause]...
statement

$omp atomic [clause[ , ] clause]...
statement

...!

C/C++ content | Fortran or For Fortran content | [n.n.n] Sections in 5.2. spec | [n.n.n] Sections in 5.1. spec | See Clause info on pg. 9

© 2021 OpenMP ARB  OMPSC21
Directives and Constructs (continued)

distribute parallel do simd and
distribute parallel for simd
[2.16.1.4]
Specifies a loop that can be executed concurrently using SIMD instructions in parallel by multiple threads that are members of multiple teams.

```c
#pragma omp distribute parallel for simd [clause[ ]',[ ,clause] ... ] loop-nest
```
clause: Any clause used for distribute, parallel for simd, or parallel do simd.

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

```c
#pragma omp taskloop simd [clause[ ]',[ ,clause] ... ] loop-nest
```
clause: Any clause used for parallel or taskloop.

parallel do and parallel for
[2.16.1]
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
```
clause: Any clause used for parallel or do except the nowait clause.

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

```c
#pragma omp parallel loop [clause[ ]',[ ,clause] ... ] loop-nest
```
clause: Any clause used for parallel or loop.

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

```c
#pragma omp parallel sections [clause[ ]',[ ,clause] ... ]
{ }
```
clause: Any clause used for parallel or sections except the nowait clause.

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

```c
#pragma omp parallel workshare [clause[ ]',[ ,clause] ... ]
```
clause: Any clause used for parallel.

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

```c
#pragma omp parallel masked taskloop simd [clause[ ]',[ ,clause] ... ] loop-nest
```
clause: Any clause used for masked taskloop simd.

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

```c
#pragma omp teams distribute [clause[ ]',[ ,clause] ... ] loop-nest
```
clause: Any clause used for teams distribute.

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

```c
#pragma omp teams distribute simd [clause[ ]',[ ,clause] ... ] loop-nest
```
clause: Any clause used for teams distribute simd.

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

```c
#pragma omp teams distribute parallel do [clause[ ]',[ ,clause] ... ] loop-nest
```
clause: Any clause used for teams distribute parallel do.

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

```c
#pragma omp teams loop [clause[ ]',[ ,clause] ... ] loop-nest
```
clause: Any clause used for teams loop.
Directives and Constructs (continued)

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

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

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

|!$omp end target parallel
```

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

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

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

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

|!$omp end target parallel do
```

clause: Any clause used for target, parallel for, or parallel do, except for copyin.

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

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

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

|!$omp end target parallel do simd
```

clause: Any clause used for target, parallel for simd, or parallel do simd, except for copyin.

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

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

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

|!$omp end target parallel loop
```

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

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

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

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

|!$omp end target simd
```

clause: Any clause used for target or simd.

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

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

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

|!$omp end target teams
```

clause: Any clause used for target or teams.

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

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

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

|!$omp end target teams distribute parallel do
```

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

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

```
#pragma omp target teams distribute parallel do simd [clause[ [, ]clause ] ...

|$omp target teams distribute parallel do simd & [clause[ [, ]clause ] ...

|!$omp end target teams distribute parallel do simd
```

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

Notes

© 2021 OpenMP ARB

www.openmp.org

OMPI21

[17] Directives: C and C++ support an additional set of clauses for target constructs. These clauses are used to specify loop-nest constructs and no other statements. The clauses are as follows: target parallel, target parallel do, target parallel for, target parallel do simd, target parallel loop, target simd, target teams, target teams parallel do, target teams parallel for, target teams simd, target teams distributed parallel do, target teams distributed parallel for, and target teams distributed parallel do simd. Each clause has a corresponding construct that must be enclosed within curly braces, and no other statements can be included outside of the curly braces. For example, the target parallel construct is specified using the clause: ```#pragma omp target parallel [clause[ [, ]clause ] ...
```

[2.16.16] Fortran content     |    C/C++ content     |    Fortran content     |    C/C++ content
```

[2.16.17] See Clause info on pg. 9
**Clauses**

All list items appearing in a clause must be visible according to the scoping rules of the base language.

**Data sharing attribute clauses**

Additional data sharing attribute clauses are `is_device_ptr`, `use_device_ptr`, `has_device_addr`, and `use_device_addr`. These clauses are described at the directives that accept them.

```text
default [shared | firstprivate | private | none]
```

Default data-sharing attributes are disabled. All variables in a construct must be declared inside the construct or appear in a data-sharing attribute clause.

```text
shared [list]
```

Variables in list are shared between threads or explicit tasks executing the construct.

```text
private [list]
```

Creates a new variable for each item in list that is private to each thread or explicit task. The private variable is not given an initial value.

```text
firstprivate [list]
```

Declares list items to be private to each thread or explicit task and assigns them the value the original variable has at the time the construct is encountered.

```text
lastprivate [(/lastprivate-modifier [list])]
```

After the last loop ends, the variables in list will be copied to the primary thread.

```text
lastprivate-modifier: conditional
```

Uses the value from the thread that executed the highest index iteration number.

```text
linear [linear-list [: | linear-step]]
```

Declares each linear-list item to have a linear value or address with respect to the iteration space of the loop.

```text
linear-list: list (or for declare simd argument-list)
```

```text
linear-modifier: step(linear-step), linear-type-modifier
```

```text
linear-step: OpenMP integer expression (1 is default)
```

```text
linear-type-modifier: val, ref, uval (val is default)
```

```text
val: The value is linear
```

```text
ref: The address is linear (C++ and Fortran only)
```

```text
uval: The value is linear, may not be modified (C++ and Fortran only)
```

The ref and uval modifiers may only be specified for a linear clause on the declare simd directive, and only for arguments that are passed by reference.

```text
allocate clause [6.6] [2.13.4]
```

```text
allocate [allocation | list]
```

```text
allocate-modifier [/, allocate-modifier | ] list
```

```text
allocate-modifier: allocate
```

```text
allocate: is an expression of:
```

```text
C++: type omp_allocate_handle_t
```

```text
alignment: A constant positive integer power of 2.
```

```text
if clause [3.4] [2.18]
```

The effect of the if clause depends on the construct to which it is applied. For combined or composite constructs, it only applies to the semantics of the construct named in the directive-name-modifier if one is specified. If no modifier is specified for a combined or composite construct then if the clause applies to all constructs to which an if clause can apply.

```text
if [directive-name-modifier | omp-logical-expression]
```

```text
map clause [5.3.8] [2.21.7.1]
```

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

Maps data from the task's environment to the device environment.

```text
order clause [10.3] [2.11.3]
```

```text
order: (order-modifier | concurrent)
```

```text
order-modifier: reproducible, unconstrained
```

```text
nowait clause [15.6]
```

```text
nowait
```

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

```text
reduction clause [5.5.8] [2.21.5.4]
```

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

```text
reduction-identifier: list
```

```text
reduction-modifier: iscan, task, default
```

- `reduction-identifier`:
  - Either an id-expression or one of the following operators: +, -, |, &, ^, &^, |&

- `c-reduction-identifier`:
  - Either an id-expression or one of the following operators: +, -, |, &, ^, &^, |&

```text
iterator clause [3.2.6] [2.16.1]
```

```text
iterator: [iterator-[type-identifier] | iterator-specifier | iterators-definition]
```

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

```text
iterator: [iterator-type] identifier = range-specification
```

```text
range-specification: begin: end: [step]
```

_begin: End: Expressions for which their types can be converted to iterator-type step: An integral expression.

```text
iterator-type: C++: A type name. For: A type specifier.
```
### Runtime Library Routines

#### Thread team routines

- **omp_set_num_threads** ([18.2.1])
  - Affects the number of threads used for subsequent parallel constructs not specifying a `num_threads` clause, by setting the value of the first element of the `num_threads` array. ICV:
    ```
    void omp_set_num_threads (int num_threads);
    ```

- **omp_get_num_threads** ([18.2.2])
  - 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. ICV:
    ```
    int omp_get_num_threads (void);
    ```

- **omp_get_max_threads** ([18.2.3])
  - 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 resumes from this routine.
    ```
    int omp_get_max_threads (void);
    ```

- **omp_get_thread_num** ([18.2.4])
  - Returns the number of the calling thread, within the current team.
    ```
    int omp_get_thread_num (void);
    ```

- **omp_in_parallel** ([18.2.5])
  - Returns true if the active-levels ICV is greater than zero; otherwise it returns false.
    ```
    int omp_in_parallel (void);
    ```

- **omp_set_dynamic** ([18.2.6])
  - Enables or disables dynamic adjustment of the number of threads available for the execution of subsequent parallel regions by setting the value of the `omp_sched_kind` ICV.
    ```
    void omp_set_dynamic (int dynamic_threads);
    ```

- **omp_get_dynamic** ([18.2.7])
  - Returns true if dynamic adjustment of the number of threads is enabled for the current task. ICV: `omp_sched_kind`.
    ```
    int omp_get_dynamic (void);
    ```

- **omp_get_cancellation** ([18.2.8])
  - Returns true if cancellation is enabled; otherwise it returns false. ICV: `omp_cancel_var`.
    ```
    int omp_get_cancellation (void);
    ```

#### omp_set_schedule ([18.2.11])

Affects the schedule that is applied when runtime is used as schedule kind, by setting the value of the `omp_sched_var` ICV.

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

#### omp_get_schedule ([18.2.12])

Returns the schedule applied when runtime schedule is used. ICV: `omp_sched_var`.

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

#### omp_get_thread_limit ([18.2.13])

Returns the number of active levels of parallelism supported.

```c
int omp_get_thread_limit ( void );
```

#### omp_set_max_active_levels ([18.2.14])

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 ICV.

```c
void omp_set_max_active_levels (int max-active-levels);
```

#### omp_get_active_level ([18.2.15])

Returns the number of active levels of parallelism supported.

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

#### omp_set_max_active_levels ([18.2.16])

Limits the number of nested active parallel regions when the innermost parallel region is generated by the current task. ICV: `omp_sched_var`.

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

#### omp_get_thread_limit ([18.2.17])

Returns the number of active levels of parallelism supported.

```c
int omp_get_thread_limit ( void );
```

#### omp_get_num_places ([18.3.2])

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

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

#### omp_get_team_size ([18.3.3])

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

```c
int omp_get_team_size ( int place_num );
```

#### omp_get_proctid ([18.3.4])

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

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

#### omp_get_place_num_procs ([18.3.5])

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

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

---

*Continued*
Runtime Library Routines (continued)

**omp_get_partition_num_places** [18.3.6] [3.3.6]
Returns the number of places in the place-partition-var ICV of the innermost implicit task.

```fortran
int omp_get_partition_num_places (void);
```

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

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

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

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

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

```fortran
size_t omp_get_thread_num (char *buffer, size_t size);
```

**omp_display_affinity** [18.3.10] [3.3.10]
Prints the OpenMP thread affinity information into a buffer.

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

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

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

Teams region routines

**omp_get_num_teams** [18.4.1] [3.4.1]
Returns the number of initial teams in the current teams region.

```fortran
int omp_get_num_teams (void);
```

**omp_get_team_num** [18.4.2] [3.4.2]
Returns the initial team number of the calling thread.

```fortran
int omp_get_team_num (void);
```

**omp_set_num_teams** [18.4.3] [3.4.3]
Sets the value of the nteams-var ICV of the current device, affecting the number of threads to be used for subsequent teams regions that do not specify a nteams-var clause.

```fortran
void omp_set_num_teams (int num_teams);
```

**omp_get_max_teams** [18.4.4] [3.4.4]
Returns an upper bound on the number of teams that could be created by a teams construct without a nteams-var clause that is encountered after execution returns from this runtime. ICV: nteams-var

```fortran
int omp_get_max_teams (void);
```

**omp_set_teams_thread_limit** [18.4.5] [3.4.5]
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.

```fortran
void omp_set_teams_thread_limit(int thread_limit);
```

**omp_get_teams_thread_limit** [18.4.6] [3.4.6]
Returns the maximum number of OpenMP threads available to participate in each contention group created by a teams construct.

```fortran
int omp_get_teams_thread_limit (void);
```

**Tasking routines**

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

```fortran
int omp_get_max_task_priority (void);
```

**omp_in_final** [18.5.3] [3.6.2]
Returns true if the current task is executing on the host device; otherwise, it returns false.

```fortran
int omp_in_final (void);
```

**Resource relinquishing routines**

**omp_pause_resource_all** [18.6.1] [3.7.1]
Allows the runtime to relinquish resources used by the current task.

```fortran
type(c_ptr) function omp_pause_resource_all( device_ptr *device_num);
```

**omp_pause_resource** [18.6.2] [3.7.1]
Allows the runtime to relinquish resources used by the current task.

```fortran
type(c_ptr) function omp_pause_resource ( device_ptr *device_num);
```

**Device memory routines**

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

**omp_target_alloc** [18.8.1] [3.8.2]
Allocates memory in a device data environment and returns a device pointer to that memory.

```fortran
void *omp_target_alloc(size_t size, int device_num);
```

**omp_target_free** [18.8.2] [3.8.2]
Frees the device memory allocated by the target alloc.

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

© 2021 OpenMP ARB  OMPSC21
**Runtime Library Routines (continued)**

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

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

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

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

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

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

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

```c
int omp_target_memcpy_async (void *dst, const void *src, size_t t_element_size, int num_dims, const void *volume, size_t t_dim_offsets, const size_t *src_device_num, const size_t *dst_device_num, int num_device_ptr, int depobj_list);
```

**omp_target_memcpy_rect** \[18.8.6\] \[3.8.6\]
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 t_element_size, int num_dims, const size_t *volume, const size_t *src_device_num, const size_t *src_dimensions, const size_t *src_offsets, int num_device_ptr, int depobj_list);
```

**omp_target_associate_ptr** \[18.8.9\] \[3.8.9\]
Maps a device pointer, which may be returned from `omp_target_memcpy()` routines, to a host pointer.

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

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

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

**omp_get_mapped_ptr** \[18.8.11\] \[3.8.11\]
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** \[18.9.1\] \[3.9.1\]

```c
void omp_init_lock (omp_lock_t *lock);
```

```c
void omp_init_lock (omp_lock_t *lock);
```

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

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

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

**Destroy lock** \[18.9.3\] \[3.9.3\]

Ensure that the OpenMP lock is uninitialized.

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

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

**Set lock** \[18.9.4\] \[3.9.4\]

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

```c
void omp_set_lock (omp_lock_t *lock);
```

```c
void omp_set_lock (omp_lock_t *lock);
```

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

```c
void omp_unset_lock (omp_lock_t *lock);
```

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

Test lock [18.9.6] [3.9.6]

Attempt to set an OpenMP lock but do not suspend execution of the task during the routine.

```c
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.

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

Memory management routines

Memory Management Types [18.13.1] [3.13.1]

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

```c
C/C++
enum omp_alloctrait_key_t
For int integer(omp_allocator_handle_kind), value ::
```

```fortran
Fortran
integer(omp_allocator_handle_kind), value ::
```

```c
C/C++
int omp_get_num_interop_properties (omp_interop_t *interop);
```

```fortran
Fortran
& subroutine omp_get_num_interop_properties (integer(kind=omp_interop_t_traits(*)) intent(in) :: interop)
```

Interoperability routines

`omp_get_num_interop_properties [18.12.1] [3.12.1]`

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

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

```fortran
Fortran
& subroutine omp_get_num_interop_properties (integer(kind=omp_interop_traits(*)) intent(in) :: interop)
```

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

```fortran
Fortran
& subroutine omp_get_interop_int (integer(kind=omp_interop_traits(*)) intent(in) :: interop, integer(* ret_code));
```

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

```fortran
Fortran
& subroutine omp_get_interop_ptr (integer(kind=omp_interop_traits(*)) intent(in) :: interop, integer(* ret_code));
```

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

```fortran
Fortran
& subroutine omp_get_interop_str (integer(kind=omp_interop_traits(*)) intent(in) :: interop, integer(* ret_code));
```

```c
C/C++
omp_get_interop_name [18.12.5] [3.12.5]
Retrieves a property name from an `omp_interop_t` object.

```fortran
Fortran
& subroutine omp_get_interop_name (integer(kind=omp_interop_traits(*)) intent(in) :: interop, integer(* ret_code));
```

```c
C/C++
omp_get_interop_type_desc [18.12.6] [3.12.6]
Retrieves a description of the type of a property associated with an `omp_interop_t` object.

```fortran
Fortran
& subroutine omp_get_interop_type_desc (integer(kind=omp_interop_traits(*)) intent(in) :: interop, integer(* ret_code));
```

```c
C/C++
omp_get_interop_rc_desc [18.12.7] [3.12.7]
Retrieves a description of the return code associated with an `omp_interop_t` object.

```fortran
Fortran
& subroutine omp_get_interop_rc_desc (integer(kind=omp_interop_traits(*)) intent(in) :: interop, integer(* ret_code));
```

```c
C/C++
omp_alloc [18.13.6] [3.13.6]
Request a memory allocation from a memory allocator.

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

```c
C/C++
omp_aligned_alloc [18.12.5]
Requests an aligned memory allocation.

```fortran
Fortran
type(c_ptr) function omp_aligned_alloc (size_t, align, allocator) bind(c) use, intrinsic :: iso_c_binding, only : c_ptr, c_size_t integer(c_size_t), value :: size integer(omp_allocator_handle_kind), value :: allocator
```

```c
C/C++
omp_calloc [18.12.7]
Requests a blocked, interleaved memory allocation.

```fortran
Fortran
type(c_ptr) function omp_calloc (size_t, nmemb, allocator) bind(c) use, intrinsic :: iso_c_binding, only : c_ptr, c_size_t integer(c_size_t), value :: size integer(omp_allocator_handle_kind), value :: allocator
```

```c
C/C++
omp_destroy_allocator [18.13.7]
Deallocates previously allocated memory.

```fortran
Fortran
void omp_destroy_allocator (ptr, allocator = omp_null_allocator);
```

```c
C/C++
omp_aligned_free [18.12.5]
Fulfills an aligned memory-free function.

```fortran
Fortran
void omp_aligned_free (ptr, allocator = omp_null_allocator);
```

```c
C/C++
omp_free [18.13.7]
Fulfills a memory-free function.

```fortran
Fortran
void omp_free (void *ptr, allocator = omp_null_allocator);
```

```c
C/C++
omp_callloc [18.13.8]
Requests a zero-initialized memory allocation from a memory allocator.

```fortran
Fortran
type(c_ptr) function omp_callloc (nmem, align, allocator) bind(c) use, intrinsic :: iso_c_binding, only : c_ptr, c_size_t integer(c_size_t), value :: nmem integer(omp_allocator_handle_kind), value :: allocator
```

Continued
Runtime Library Routines (continued)

**omp_realloc** [18.13.9] [13.9]
Reallocates the given area of memory originally allocated by `free_allocator` using allocator, moving and resizing if necessary.

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

**Environment Variables**

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

### OMP_AFFINITY_FORMAT format [21.2.5] [6.14]
Sets the initial value of 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 `\%[n][0-9].size_type`, where the field type may be either the short or long names for an integer field specifier.

```
setenv OMP_AFFINITY_FORMAT \\n  \%10.4i_\%12.4i_\%6.2f_\%2.2f_\%1.2f_\%6.2f
```

**OMP_ALLOCATOR** [21.5.1] [6.22]
OpenMP memory allocators can be used to make allocator 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 6.1.2.8
- **Allocator traits** are listed in Table 6.2.2.9
- **Predefined allocators** are listed in Table 6.3.10

### Examples

```
setenv OMP_ALLOCATOR omp_default_mem_space
```

### Memory space names

- `omp_default_mem_space`
- `omp_large_mem_space`
- `omp_const_mem_space`

### Tool control routine

**omp_control_tool** [18.14] [3.14]
Enables a program to pass commands to an active tool.

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

### Environment display routine

**omp_display_env** [18.15] [3.15]
Displays the OpenMP version number and the values of ICVs associated with environment variables.

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

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

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

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

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

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

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

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

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

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

OMP_WAIT_POLICY [policy] [21.2.3] [6.7]
Sets the wait-policy-var ICV that provides a hint to an OpenMP implementation about the desired behavior of waiting threads. Valid values for policy are active (waiting threads consume processor cycles while waiting) and passive. Default is implementation defined.

Internal Control Variables (ICV) Values

Host and target device ICVs are initialized before OpenMP API constructs or routines execute. After initial values are assigned, the values of environment variables set by the user are read and the associated ICVs for host and target devices are modified accordingly. Certain environment variables may be extended with device-specific environment variables with the following syntax: <ENV_VAR>.DEV<device_name>. 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 2.1-3) [2.1-3]

<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>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>affinity-format-var</td>
<td>OMP_AFFINITY_FORMAT</td>
<td>Implementation defined.</td>
<td>omp_set_affinity_format()</td>
<td>omp_get_affinity_format()</td>
<td>Device</td>
<td>[21.2.5] [6.14]</td>
</tr>
<tr>
<td>bind-var</td>
<td>OMP_PROC_BIND</td>
<td>Implementation defined.</td>
<td>(none)</td>
<td>omp_get_proc_bind()</td>
<td>Data env.</td>
<td>[21.1.7] [6.4]</td>
</tr>
<tr>
<td>cancel-var</td>
<td>OMP_CANCELLATION</td>
<td>false</td>
<td>(none)</td>
<td>omp_get_cancellation()</td>
<td>Global</td>
<td>[21.2.6] [6.11]</td>
</tr>
<tr>
<td>debug-var</td>
<td>OMP_DEBUG</td>
<td>disabled</td>
<td>(none)</td>
<td>(none)</td>
<td>(none)</td>
<td>[21.1.6] [6.21]</td>
</tr>
<tr>
<td>def-locator-var</td>
<td>OMP_ALLOCATOR</td>
<td>Implementation defined.</td>
<td>omp_set_default_locator()</td>
<td>omp_get_default_locator()</td>
<td>Impl. Task</td>
<td>[21.2.5] [6.12]</td>
</tr>
<tr>
<td>def-sched-var</td>
<td></td>
<td>(none)</td>
<td>(none)</td>
<td>(none)</td>
<td>Device</td>
<td>---</td>
</tr>
<tr>
<td>default-device-var</td>
<td>OMP_DEFAULT_DEVICE</td>
<td>Implementation defined.</td>
<td>omp_set_default_device()</td>
<td>omp_get_default_device()</td>
<td>Data env.</td>
<td>[21.2.6] [6.15]</td>
</tr>
<tr>
<td>display-affinity-var</td>
<td>OMP_DISPLAY_AFFINITY</td>
<td>false</td>
<td>(none)</td>
<td>(none)</td>
<td>Global</td>
<td>[21.2.6] [6.13]</td>
</tr>
<tr>
<td>dyn-var</td>
<td>OMP_DYNAMIC</td>
<td>Implementation-defined if the implementation supports dynamic adjustment of the number of threads; otherwise, the initial value is false.</td>
<td>omp_set_dynamic()</td>
<td>omp_get_dynamic()</td>
<td>Data env.</td>
<td>[21.1.1] [6.3]</td>
</tr>
<tr>
<td>explicit-task-var</td>
<td>(none)</td>
<td>false</td>
<td>(none)</td>
<td>(none)</td>
<td>omp_in_explicit_task()</td>
<td>Data env.</td>
</tr>
<tr>
<td>final-task-var</td>
<td>(none)</td>
<td>false</td>
<td>(none)</td>
<td>(none)</td>
<td>omp_in_final()</td>
<td>Data env.</td>
</tr>
<tr>
<td>levels-var</td>
<td>(none)</td>
<td>zero</td>
<td>(none)</td>
<td>(none)</td>
<td>omp_get_level()</td>
<td>Data env.</td>
</tr>
<tr>
<td>max-active-levels-var</td>
<td>OMP_MAX_ACTIVE_LEVELS, OMP_NUM_THREADS, OMP_PROC_BIND</td>
<td>Implementation defined.</td>
<td>omp_set_max_active_levels()</td>
<td>omp_get_max_active_levels()</td>
<td>Device</td>
<td>[21.1.4] [6.8], [21.1.2] [6.9], [21.1.7] [6.4]</td>
</tr>
<tr>
<td>max-task-priority-var</td>
<td>OMP_MAX_TASK_PRIORITY</td>
<td>zero</td>
<td>(none)</td>
<td>(none)</td>
<td>omp_get_max_task_priority()</td>
<td>Global</td>
</tr>
<tr>
<td>nteams-var</td>
<td>OMP_NUM_TEAMS</td>
<td>zero</td>
<td>(none)</td>
<td>(none)</td>
<td>omp_get_max_teams()</td>
<td>Device env.</td>
</tr>
<tr>
<td>nthreads-var</td>
<td>OMP_NUM_THREADS</td>
<td>Implementation defined.</td>
<td>omp_set_max_threads()</td>
<td>omp_get_max_threads()</td>
<td>Data env.</td>
<td>[21.1.2] [6.2]</td>
</tr>
<tr>
<td>num-procs-var</td>
<td>(none)</td>
<td>Implementation defined.</td>
<td>(none)</td>
<td>omp_get_num_procs()</td>
<td>Device</td>
<td>---</td>
</tr>
<tr>
<td>place_partition-var</td>
<td>OMP_PLACES</td>
<td>Implementation defined.</td>
<td>(none)</td>
<td>(none)</td>
<td>omp_get_partition()</td>
<td>Impl. Task</td>
</tr>
<tr>
<td>run-sched-var</td>
<td>OMP_SCHEDULER</td>
<td>Implementation defined.</td>
<td>omp_get_schedule()</td>
<td>(none)</td>
<td>Data env.</td>
<td>[21.2.1] [6.1]</td>
</tr>
<tr>
<td>stacksize-var</td>
<td>OMP_STACKSIZE</td>
<td>Implementation defined.</td>
<td>(none)</td>
<td>(none)</td>
<td>Device</td>
<td>[21.2.2] [6.6]</td>
</tr>
<tr>
<td>target-offload-var</td>
<td>OMP_TARGET_OFFLOAD</td>
<td>DEFAULT</td>
<td>(none)</td>
<td>(none)</td>
<td>Global</td>
<td>[21.2.8] [6.17]</td>
</tr>
<tr>
<td>team-size-var</td>
<td>(none)</td>
<td>one</td>
<td>(none)</td>
<td>(none)</td>
<td>omp_get_num_threads()</td>
<td>Team</td>
</tr>
<tr>
<td>teams-thread-limit-var</td>
<td>OMP_TEAMS_THREAD_LIMIT</td>
<td>zero</td>
<td>(none)</td>
<td>(none)</td>
<td>omp_get_teams_thread_limit()</td>
<td>Data env.</td>
</tr>
<tr>
<td>thread-limit-var</td>
<td>OMP_THREAD_LIMIT</td>
<td>Implementation defined.</td>
<td>(none)</td>
<td>omp_get_teams_thread_limit()</td>
<td>Device</td>
<td>[21.2.6] [6.24]</td>
</tr>
<tr>
<td>thread-num-var</td>
<td>(none)</td>
<td>zero</td>
<td>(none)</td>
<td>(none)</td>
<td>omp_get_thread_num()</td>
<td>Impl. Task</td>
</tr>
<tr>
<td>tool-libraries-var</td>
<td>OMP_TOOL_LIBRARIES</td>
<td>empty string</td>
<td>(none)</td>
<td>(none)</td>
<td>Global</td>
<td>[21.2.13] [6.18]</td>
</tr>
<tr>
<td>tool-var</td>
<td>OMP_TOOL</td>
<td>enabled</td>
<td>(none)</td>
<td>(none)</td>
<td>Global</td>
<td>[21.2.13] [6.18]</td>
</tr>
<tr>
<td>tool-verbose-init-var</td>
<td>OMP_TOOL_VERBOSE_INIT</td>
<td>disabled</td>
<td>(none)</td>
<td>(none)</td>
<td>Global</td>
<td>[21.2.3] [6.20]</td>
</tr>
<tr>
<td>wait-policy-var</td>
<td>OMP_WAIT_POLICY</td>
<td>Implementation defined.</td>
<td>(none)</td>
<td>(none)</td>
<td>Device</td>
<td>[21.2.3] [6.17]</td>
</tr>
</tbody>
</table>
Using OpenMP Tools

A tool indicates its interest in using the OMPT interface by providing a non-null pointer to an ompt_start_tool_result_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).

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

OpenMP API 5.2 Reference Guide Index

Clauses-9

Directives and Constructs-1–8
allocate, allocators-1
assume, assumes-2
atomic-6
barrier-5
begin assumes-2
begin declare target-2
cancel-6
Cancellation constructs
cancel-6
cancellation point-6
cancellation point-6
Combined forms-6–8
critical-5
Data environment directives-1
declare mapper-1
declare reduction-1
scan-1
threadprivate-1
declare mapper-1
declare reduction-1
declare simd-2
declare target-2
declare variant-2
depobj-5
Device directives and construct-4
target-5
target data-4
target enter data-5
target exit data-5
target update-5
dispatch-2
distribute-4
do-4
flush-5
for-4
Informational & utility directives-2
assume, [begin ]assumes-2
error-3
nothing-3
requires-2
interop-5
loop-4
Loop transformation constructs-3
tile-3
unroll-3
masked-3
Memory management directives-1
allocate-1
allocators-1
Memory spaces-1
metadirective-1
nothing-3
ordered-6
parallel-3
Parallelism constructs-3
masked-3
parallel-3
simd-3
teams-3
requires-2
scan-1
scope-3
section, sections-4
simd-3
single-3
Synchronization constructs-5
atomic-6
barrier-5
critical-5
depobj-5
flush-5
ordered-6
taskgroup-5
taskwait-5
target-5
target data-4
target enter data-5
target exit data-5
target update-5
task-4
taskgroup-5
Tasking constructs-4
task-4
taskloop-4
taskyield-4
taskwait-5
taskyield-4
tasks-3
threadprivate-1
tile-3
unroll-3
Variant directives-1
[begin ]declare target-2
declare simd-2
declare variant-2
dispatch-2
metadirective-1
Work-distribution constructs-3
distribute-4
do-4
for-4
loop-4
scope-3
section, sections-4
single-3
workshare-3
workshare-3
Environment Variables-14–15
Internal Control Variable (ICV) Values-15
Runtime Library Routines-10–14
Device information routines-11
Device memory routines-11–12
Environment display routine-14
Event routine-13
Interoperability routines-13
Lock routines-12–13
Memory management routines-13
Resource relinquishing routines-11
Tasking routines-11
Teams region routines-11
Thread affinity routines-10–11
Thread team routines-10
Timing routines-13
Tool control routine-14

Copyright © 2021 OpenMP Architecture Review Board. Permission to copy without fee all or part of this material is granted, provided the OpenMP Architecture Review Board copyright notice and the title of this document appear. Notice is given that copying is by permission of the OpenMP Architecture Review Board. Portions of this product/publication may have been derived from the OpenMP Language Application Program Interface Specification.