OpenMP®

Fortran

OpenMP 4.5 API Fortran Syntax Reference Guide

OpenMP Application Program Interface (API) is a portable, scalable model that gives parallel programmers a simple and flexible interface for developing portable parallel applications. OpenMP supports multi-platform shared-memory parallel programming in C/C++ and Fortran on all architectures, including Unix platforms and Windows platforms. See www.openmp.org for specifications.

- Text in this color indicates functionality that is new or changed in the OpenMP API 4.5 specification.
- [n.n.n] Refers to sections in the OpenMP API 4.5 specification.
- [n.n.n] Refers to sections in the OpenMP API 4.0 specification.

Directives and Constructs for Fortran

OpenMP directives are specified in Fortran by using special comments that are identified by unique sentinels. Also, a special comment form is available for conditional Fortran compilation. An OpenMP executable directive applies to the succeeding structured block. A *structured-block* is a block of executable statements with a single entry at the top and a single exit at the bottom, or an OpenMP construct. OpenMP directives except SIMD and **declare target** directives may not appear in **PURE** or **ELEMENTAL** procedures.

parallel [2.5] [2.5]

Forms a team of threads and starts parallel execution.

!$omp parallel *[clause[[,]clause] ...]*
  structured-block
!$omp end parallel

clause:

if(*[***parallel :***] scalar-logical-expression*)
num_threads(*scalar-integer-expression*)
default(**private** | **firstprivate** | **shared** | **none**)
private(*list*)
firstprivate(*list*)
shared(*list*)
copyin(*list*)
reduction(*reduction-identifier* **:** *list*)
proc_bind(**master** | **close** | **spread**)

do [2.7.1] [2.5.1]

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

!$omp do *[clause[[,]clause] ...]*
  do-loops
*[***!$omp end do** *[***nowait***]]*

clause:

private(*list*)
firstprivate(*list*)
lastprivate(*list*)
linear(*list[: linear-step]*)
reduction(*reduction-identifier* **:** *list*)
schedule(*[modifier [, modifier]* **:** *]* *kind[*, *chunk_size]*)
collapse(*n*)
ordered *[*(*n*)*]*

kind:

- **static:** Iterations are divided into chunks of size *chunk_size* and assigned to threads in the team in round-robin fashion in order of thread number.
- **dynamic:** Each thread executes a chunk of iterations then requests another chunk until no chunks remain to be distributed.
- **guided:** Each thread executes a chunk of iterations then requests another chunk until no chunks remain to be assigned.
- **auto:** The decision regarding scheduling is delegated to the compiler and/or runtime system.
- **runtime:** The schedule and chunk size are taken at runtime from the *run-sched-var* ICV.

modifier:

- **monotonic:** Each thread executes the chunks that it is assigned in increasing logical iteration order.
- **nonmonotonic:** Chunks are assigned to threads in any order and the behavior of an application that depends on execution order of the chunks is unspecified.
- **simd:** Ignored when the loop is not associated with a SIMD construct, otherwise the *new_chunk_size* for all except the first and last chunks is ⌈*chunk_size/simd_width*⌉ ∗ *simd_width* where *simd_width* is an implementation-defined value.

sections [2.7.2] [2.5.2]

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

!$omp sections *[clause[[,] clause] ...]*
  *[***!$omp section***]*
    structured-block
  *[***!$omp section**
    structured-block]
  ...
!$omp end sections *[***nowait***]*

clause:

private(*list*)
firstprivate(*list*)
lastprivate(*list*)
reduction(*reduction-identifier* **:** *list*)

single [2.7.3] [2.7.3]

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

!$omp single *[clause[[,]clause] ...]*
  structured-block
!$omp end single *[end_clause[[,]end_clause] ...]*

clause:

private(*list*)
firstprivate(*list*)

end_clause:

copyprivate(*list*)
nowait

workshare [2.7.4] [2.7.4]

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

!$omp workshare
  structured-block
!$omp end workshare *[***nowait***]*

The structured block must consist of only the following:

array or scalar assignments
FORALL or **WHERE** statements
WHERE, **atomic**, **critical**, or **parallel** constructs

simd [2.8.1] [2.8.1]

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

!$omp simd *[clause[[,]clause] ...]*
  do-loops
*[***!$omp end simd***]*

clause:

safelen(*length*)
simdlen(*length*)
linear(*list[* **:** *linear-step]*)
aligned(*list[* **:** *alignment]*)
private(*list*)
lastprivate(*list*)
reduction(*reduction-identifier* **:** *list*)
collapse(*n*)

declare simd [2.8.2] [2.8.2]

Applied to a function or a subroutine to enable the creation of one or more versions that can process multiple arguments using SIMD instructions from a single invocation from a SIMD loop.

!$omp declare simd *[*(*proc-name*)*]* *[clause[[,]clause] ...]*

clause:

simdlen(*length*)
linear(*linear-list[* **:** *linear-step]*)
aligned(*argument-list[* **:** *alignment]*)
uniform(*argument-list*)
inbranch
notinbranch

do simd [2.8.3] [2.8.3]

Specifies a loop that can be executed concurrently using SIMD instructions and that those iterations will also be executed in parallel by threads in the team.

!$omp do simd *[clause[[,]clause] ...]*
  do-loops
*[***!$omp end do simd** *[***nowait***]]*

clause:

Any accepted by the **simd** or **do** directives with identical meanings and restrictions.

task [2.9.1] [2.11.1]

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

!$omp task *[clause[[,]clause] ...]*
  structured-block
!$omp end task

clause may be:

if(*[* **task :** *] scalar-logical-expression*)
final(*scalar-logical-expression*)
untied
default(**private** | **firstprivate** | **shared** | **none**)
mergeable
private(*list*)
firstprivate(*list*)
shared(*list*)
depend(*dependence-type* **:** *list*)
priority(*priority-value*)

Continued

Directives and Constructs for Fortran (continued)

taskloop [2.9.2]

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

```
!$omp taskloop [clause[ [, ]clause] ...]
    do-loops
[!$omp end taskloop ]
clause:
    if([ taskloop : ] scalar-logical-expression)
    shared(list)
    private(list)
    firstprivate(list)
    lastprivate(list)
    default(private | firstprivate | shared | none)
    grainsize(grain-size)
    num_tasks(num-tasks)
    collapse(n)
    final(scalar-logical-expression)
    priority(priority-value)
    untied
    mergeable
    nogroup
```

taskloop simd [2.9.3]

Specifies that a loop that can be executed concurrently using SIMD instructions, and that those iterations will also be executed in parallel using OpenMP tasks.

```
!$omp taskloop simd [clause[ [, ]clause] ...]
    do-loops
[!$omp end taskloop simd]
```

clause:
Any accepted by the **simd** or **taskloop** directives with identical meanings and restrictions.

taskyield [2.9.4] [2.11.2]

Specifies that the current task can be suspended in favor of execution of a different task.

```
!$omp taskyield
```

target data [2.10.1] [2.9.1]

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

```
!$omp target data clause[ [ [, ]clause] ...]
structured-block
!$omp end target data
clause:
    if( [target data : ] scalar-logical-expression)
    device(scalar-integer-expression)
    map( [[map-type-modifier[,]] map-type : ] list)
    use_device_ptr(list)
```

target enter data [2.10.2]

Specifies that variables are mapped to a device data environment.

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

clause:
    if([target enter data : ] scalar-logical-expression)
    device(scalar-integer-expression)
    map( [[map-type-modifier[,]] map-type : ] list)
    depend(dependence-type : list)
    nowait
```

target exit data [2.10.3]

Specifies that list items are unmapped from a device data environment.

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

clause:
    if([target exit data : ] scalar-logical-expression)
    device(scalar-integer-expression)
    map( [[map-type-modifier[,]] map-type : ] list)
    depend(dependence-type : list)
    nowait
```

target [2.10.4] [2.9.2]

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

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

clause:
    if([target  : ] scalar-logical-expression)
    device(scalar-integer-expression)
    private(list)
    firstprivate(list)
    map( [[map-type-modifier[,]] map-type : ] list)
    is_device_ptr(list)
    defaultmap(tofrom : scalar)
    nowait
    depend(dependence-type : list)
```

target update [2.10.5] [2.9.3]

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

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

clause: motion-clause or one of:
    if([target update  : ] scalar-logical-expression)
    device(scalar-integer-expression)
    nowait
    depend(dependence-type : list)

motion-clause:
    to(list)
    from(list)
```

declare target [2.10.6] [2.9.4]

A declarative directive that specifies that variables and functions are mapped to a device.

```
!$omp declare target  [clause[ [, ]clause] ...]

clause:
    to(extended-list)
    link(list)
```

```
!$omp declare target (extended-list)
```

extended-list: A comma-separated list of named variables, procedure names, and named common blocks.

teams [2.10.7] [2.9.5]

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

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

clause:
    num_teams(scalar-integer-expression)
    thread_limit(scalar-integer-expression)
    default(shared | firstprivate | private | none)
    private(list)
    firstprivate(list)
    shared(list)
    reduction(reduction-identifier : list)
```

distribute [2.10.8] [2.9.6]

Specifies loops which are executed by the thread teams.

```
!$omp distribute [clause[ [, ]clause] ...]
    do-loops
[!$omp end distribute]

clause:
    private(list)
    firstprivate(list)
    lastprivate(list)
    collapse(n)
    dist_schedule(kind[, chunk_size])
```

distribute simd [2.10.9] [2.9.7]

Specifies loops which are executed concurrently using SIMD instructions.

```
!$omp distribute simd [clause[ [, ]clause] ...]
    do-loops
[!$omp end distribute simd]
```

clause: Any of the clauses accepted by **distribute** or **simd**.

distribute parallel do [2.10.10] [2.9.8]

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

```
!$omp distribute parallel do [clause[ [, ]clause] ...]
    do-loops
[!$omp end distribute parallel do]
```

clause: Any accepted by the **distribute** or **parallel do** directives.

distribute parallel do simd [2.10.11] [2.9.9]

These constructs specify a loop that can be executed in parallel using SIMD semantics in the simd case by multiple threads that are members of multiple teams.

```
!$omp distribute parallel do simd [clause[ [, ]clause] ...]
    do-loops
[!$omp end distribute parallel do simd]
```

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

parallel do [2.11.1] [2.10.1]

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

```
!$omp parallel do [clause[ [, ]clause] ...]
    do-loops
[!$omp end parallel do]
```

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

parallel sections [2.11.2] [2.10.2]

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

```
!$omp parallel sections [clause[ [, ]clause] ...]
    [!$omp section]
        structured-block
    [!$omp section
        structured-block]
    ...
!$omp end parallel sections
```

clause: Any of the clauses accepted by the **parallel** or **sections** directives, with identical meanings and restrictions. The last section ends at the **end parallel sections** directive. **nowait** cannot be specified on an **end parallel sections** directive.

parallel workshare [2.11.3] [2.10.3]

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

```
!$omp parallel workshare [clause[ [, ]clause] ...]
    structured-block
!$omp end parallel workshare
```

clause: Any of the clauses accepted by the **parallel** directive, with identical meanings and restrictions.

Continued

Directives and Constructs for Fortran (continued)

parallel do simd [2.11.4] [2.10.4]

Shortcut for specifying a **parallel** construct containing one **do simd** construct and no other statements.

!$omp parallel do simd *[clause[[,]clause] ...]*
do-loops
[!$omp end parallel do simd]

clause: Any accepted by the **parallel** or **do simd** directives with identical meanings and restrictions. If an **end parallel do simd** directive is not specified, then an **end parallel do simd** directive is assumed at the end of the *do-loops*.

target parallel [2.11.5]

Shortcut for specifying a **target** construct containing a **parallel** construct and no other statements.

!$omp target parallel *[clause[[,]clause] ...]*
structured-block
!$omp end target parallel

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

target parallel do [2.11.6]

Shortcut for specifying a **target** construct containing a **parallel do** construct and no other statements.

!$omp target parallel do *[clause[[,]clause] ...]*
do-loops
*[***!$omp end target parallel do***]*

clause: Any accepted by the **target** or **parallel do** directives, except for **copyin**, with identical meanings and restrictions.

target parallel do simd [2.11.7]

Shortcut for specifying a **target** construct containing a **parallel do simd** construct and no other statements.

!$omp target parallel do simd *[clause[[,]clause] ...]*
do-loops
*[***!$omp end target parallel do simd***]*

clause: Any accepted by the **target** or **parallel do simd** directives, except for **copyin**, with identical meanings and restrictions.

target simd [2.11.8]

Shortcut for specifying a **target** construct containing a **simd** construct and no other statements.

!$omp target simd *[clause[[,]clause] ...]*
do-loops
*[***!$omp end target simd***]*

clause: Any accepted by the **target** or **simd** directives with identical meanings and restrictions.

target teams [2.11.9] [2.10.5]

Shortcut for specifying a **target** construct containing a **teams** construct and no other statements.

!$omp target teams *[clause[[,]clause] ...]*
structured-block
!$omp end target teams

clause:
Any accepted by the **target** or **teams** directives with identical meanings and restrictions.

teams distribute [2.11.10] [2.10.6]

Shortcuts for specifying **teams** constructs containing a **distribute** construct and no other statements.

!$omp teams distribute *[clause[[,]clause] ...]*
do-loops
[**!$omp end teams distribute** *]*

clause: Any accepted by the **teams** or **distribute** directives with identical meanings and restrictions.

teams distribute simd [2.11.11] [2.10.7]

Shortcuts for specifying **teams** constructs containing a **distribute simd** construct and no other statements.

!$omp teams distribute simd *[clause[[,]clause] ...]*
do-loops
[**!$omp end teams distribute simd** *]*

clause: Any accepted by the **teams** or **distribute simd** directives with identical meanings and restrictions.

target teams distribute [2.11.12] [2.10.8]

Shortcuts for specifying a **target** construct containing a **teams distribute** construct and no other statements.

!$omp target teams distribute *[clause[[,]clause] ...]*
do-loops
[**!$omp end target teams distribute** *]*

clause: Any accepted by the **target** or **teams distribute** directives with identical meanings and restrictions.

target teams distribute simd [2.11.13] [2.10.9]

Shortcuts for specifying a **target** construct containing a **teams distribute simd** construct and no other statements.

!$omp target teams distribute simd *[clause[[,]clause] ...]*
do-loops
[**!$omp end target teams distribute simd** *]*

clause: Any accepted by the **target** or **teams distribute simd** directives with identical meanings and restrictions.

teams distribute parallel do [2.11.14] [2.10.10]

Shortcuts for specifying **teams** constructs containing a **distribute parallel do** construct and no other statements.

!$omp teams distribute parallel do *[clause[[,]clause] ...]*
do-loops
[**!$omp end teams distribute parallel do** *]*

clause: Any clause used for **teams** or **distribute parallel do** directives with identical meanings and restrictions.

target teams distribute parallel do [2.11.15] [2.10.11]

Shortcut for specifying a **target** construct containing a **teams distribute parallel do** construct and no other statements.

!$omp target teams distribute parallel do &
!$omp *[clause[[,]clause] ...]*
do-loops
*[***$omp end target teams distribute parallel do***]*

clause:
Any clause used for **teams distribute parallel do** or **target** directives with identical meanings and restrictions.

teams distribute parallel do simd [2.11.16] [2.10.12]

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

!$omp teams distribute parallel do simd &
!$omp *[clause[[,]clause] ...]*
do-loops
*[***!$omp end teams distribute parallel do simd***]*

clause: Any clause used for **teams** or **distribute parallel do simd** directives with identical meanings and restrictions.

target teams distribute parallel do simd [2.11.17] [2.10.13]

Shortcut for specifying a **target** construct containing a teams distribute parallel do simd construct and no other statements.

!$omp target teams distribute parallel do simd &
!$omp *[clause[[,]clause] ...]*
do-loops
*[***!$omp end target teams distribute parallel do simd***]*

clause: Any clause used for **teams distribute parallel do simd** or **target** directives with identical meanings and restrictions.

master [2.13.1] [2.12.1]

Specifies a structured block that is executed by the master thread of the team.

!$omp master
structured-block
!$omp end master

critical [2.13.2] [2.12.2]

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

!$omp critical *[***(***name***)** *[***hint(***hint-expression***)***]]*
structured-block
!$omp end critical *[***(***name***)***]*

barrier [2.13.3] [2.12.3]

Placed only at a point where a base language statement is allowed, this directive specifies an explicit barrier at the point at which the construct appears.

!$omp barrier

taskwait [2.13.4] [2.12.4]

Specifies a wait on the completion of child tasks of the current task.

!$omp taskwait

taskgroup [2.13.5] [2.12.5]

Specifies a wait on the completion of child tasks of the current task, and waits for descendant tasks.

!$omp taskgroup
structured-block
!$omp end taskgroup

atomic [2.13.6] [2.12.6]

Ensures a specific storage location is accessed atomically. May take one of the following seven forms:

!$omp atomic *[***seq_cst***[,]]* **read** *[[,]***seq_cst***]*
capture-statement
*[***!$omp end atomic***]*

!$omp atomic *[***seq_cst***[,]]* **write** *[[,]***seq_cst***]*
write-statement
*[***!$omp end atomic***]*

!$omp atomic *[***seq_cst***[,]]* **update** *[[,]***seq_cst***]*
update-statement
*[***!$omp end atomic***]*

!$omp atomic *[***seq_cst***]*
update-statement
*[***!$omp end atomic***]*

!$omp atomic *[***seq_cst***[,]]* **capture** *[[,]***seq_cst***]*
update-statement
capture-statement
!$omp end atomic

(**atomic** continues on the next page)

Continued

Directives and Constructs for Fortran (continued)

atomic (continued)

!$omp atomic *[***seq_cst***[,]]* **capture** *[[,]***seq_cst***]*
capture-statement
update-statement
!$omp end atomic

!$omp atomic *[***seq_cst***[,]]* **capture** *[[,]***seq_cst***]*
capture-statement
write-statement
!$omp end atomic

capture-stmt, *write-stmt*, or *update-stmt* may be:

capture-statement	*v = x*
write-statement	*x = expr*
update-statement	*x = x operator expr* *x = expr operator x* *x = intrinsic_procedure_name (x, expr_list)* *x = intrinsic_procedure_name (expr_list, x)*
intrinsic_procedure_name is one of **MAX**, **MIN**, **IAND**, **IOR**, **IEOR** *operator* is one of **+**, *****, **-**, **/**, **.AND.**, **.OR.**, **.EQV.**, **.NEQV.**	

flush [2.13.7] [2.12.7]

Makes a thread's temporary view of memory consistent with memory, and enforces an order on the memory operations of the variables.

!$omp flush *[***(***list***)***]*

ordered [2.13.8] [2.12.8]

Specifies a structured block in a loop, **simd**, or loop SIMD region that will be executed in the order of the loop iterations.

!$omp ordered *[clause[[,]clause] ...]*
structured-block
!$omp end ordered

clause:
threads
simd

(**ordered** continues in the next column)

ordered (continued)

!$omp ordered *clause[[[,]clause] ...]*

clause:
depend (**source**)
depend (**sink** **:** *vec*)

cancel [2.14.1] [2.13.1]

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

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

construct-type-clause:
parallel
sections
do
taskgroup

if-clause:
if(*scalar-logical-expression***)**

cancellation point [2.14.2] [2.13.2]

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*

construct-type-clause:
parallel
sections
do
taskgroup

threadprivate [2.15.2] [2.14.2]

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.

!$omp threadprivate(*list***)**

list:
A comma-separated list of named variables and named common blocks. Common block names must appear between slashes.

declare reduction [2.16] [2.15]

Declares a *reduction-identifier* that can be used in a **reduction** clause.

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

reduction-identifier:
A base language identifier, user defined operator, or one of the following operators*:*
+, -, *, .and., .or., .eqv., .negv., or one of the following intrinsic procedure names: max, min, iand, ior, ieor.

type-list: A list of type specifiers

combiner: An assignment statement or a subroutine name followed by an argument list

initializer-clause: **initializer** (*initializer-expr*) where
initializer-expr is **omp_priv** = *initializer* or
function-name (*argument-list*)

Runtime Library Routines for Fortran

Execution environment routines affect and monitor threads, processors, and the parallel environment. The library routines are external procedures.

Execution Environment Routines

omp_set_num_threads [3.2.1] [3.2.1]

Affects the number of threads used for subsequent parallel regions not specifying a **num_threads** clause, by setting the value of the first element of the *nthreads-var* ICV of the current task to *num_threads*.

subroutine omp_set_num_threads(*num_threads***)**
integer *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.

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.

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.

integer **function omp_get_thread_num()**

omp_get_num_procs [3.2.5] [3.2.5]

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

integer **function omp_get_num_procs()**

omp_in_parallel [3.2.6] [3.2.6]

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

logical **function omp_in_parallel()**

omp_set_dynamic [3.2.7] [3.2.7]

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

subroutine omp_set_dynamic(*dynamic_threads***)**
logical *dynamic_threads*

omp_get_dynamic [3.2.8] [3.2.8]

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

logical **function omp_get_dynamic()**

omp_get_cancellation [3.2.9] [3.2.9]

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

logical **function omp_get_cancellation()**

omp_set_nested [3.2.10] [3.2.10]

Enables or disables nested parallelism, by setting the *nest-var* ICV.

subroutine omp_set_nested(*nested***)**
logical *nested*

omp_get_nested [3.2.11] [3.2.11]

Returns the value of the *nest-var* ICV, which indicates if nested parallelism is enabled or disabled.

logical **function omp_get_nested()**

omp_set_schedule [3.2.12] [3.2.12]

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

subroutine omp_set_schedule(*kind*, *chunk_size***)**
integer **(kind=omp_sched_kind)** *kind*
integer *chunk_size*

See *kind* for **omp_get_schedule**.

Continued

Runtime Library Routines for Fortran (continued)

Return types are shown in green.

omp_get_schedule [3.2.13] [3.2.13]

Returns the value of *run-sched-var* ICV, which is the schedule applied when **runtime** schedule is used.

```
subroutine omp_get_schedule(kind, chunk_size)
integer (kind=omp_sched_kind) kind
integer chunk_size
```

kind for **omp_set_schedule** and **omp_get_schedule** is an implementation-defined schedule or:

omp_sched_static	= 1
omp_sched_dynamic	= 2
omp_sched_guided	= 3
omp_sched_auto	= 4

omp_get_thread_limit [3.2.14] [3.2.14]

Returns the value of the *thread-limit-var* ICV, which is the maximum number of OpenMP threads available.

```
integer function omp_get_thread_limit()
```

omp_set_max_active_levels [3.2.15] [3.2.15]

Limits the number of nested active parallel regions, by setting *max-active-levels-var* ICV.

```
subroutine omp_set_max_active_levels(max_levels)
integer max_levels
```

omp_get_max_active_levels [3.2.16] [3.2.16]

Returns the value of *max-active-levels-var* ICV, which determines the maximum number of nested active parallel regions.

```
integer function omp_get_max_active_levels()
```

omp_get_level [3.2.17] [3.2.17]

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

```
integer function omp_get_level()
```

omp_get_ancestor_thread_num [3.2.18] [3.2.18]

Returns, for a given nested level of the current thread, the thread number of the ancestor of the current thread.

```
integer function omp_get_ancestor_thread_num(level)
integer level
```

omp_get_team_size [3.2.19] [3.2.19]

Returns, for a given nested level of the current thread, the size of the thread team to which the ancestor or the current thread belongs.

```
integer function omp_get_team_size(level)
integer level
```

omp_get_active_level [3.2.20] [3.2.20]

Returns the value of the *active-level-vars* ICV, which determines the number of active, nested **parallel** regions enclosing the task that contains the call.

```
integer function omp_get_active_level()
```

omp_in_final [3.2.21] [3.2.21]

Returns *true* if the routine is executed in a final task region; otherwise, it returns *false*.

```
logical function omp_in_final()
```

omp_get_proc_bind [3.2.22] [3.2.22]

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

```
integer (kind=omp_proc_bind_kind)&
    function omp_get_proc_bind()
```

Returns one of:

omp_proc_bind_false	= 0
omp_proc_bind_true	= 1
omp_proc_bind_master	= 2
omp_proc_bind_close	= 3
omp_proc_bind_spread	= 4

omp_get_ num_places [3.2.23]

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

```
integer function omp_get_num_places()
```

omp_get_place_num_procs [3.2.24]

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

```
integer function omp_get_place_num_procs(place_num)
integer place_num
```

omp_get_place_proc_ids [3.2.25]

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

```
subroutine omp_get_place_proc_ids(place_num, ids)
integer place_num
integer ids (*)
```

omp_get_place_num [3.2.26]

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

```
integer function omp_get_place_num()
```

omp_get_partition_num_places [3.2.27]

Returns the number of places in the place partition of the innermost implicit task.

```
integer function omp_get_partition_num_places()
```

omp_get_partition_place_nums [3.2.28]

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

```
subroutine omp_get_partition_place_nums(place_nums)
integer place_nums (*)
```

omp_set_default_device [3.2.29] [3.2.23]

Assigns the value of the *default-device-var* ICV, which determines default target device.

```
subroutine omp_set_default_device(device_num)
integer device_num
```

omp_get_default_device [3.2.30] [3.2.24]

Returns the value of the *default-device-var* ICV, which determines default target device.

```
integer function omp_get_default_device()
```

omp_get_num_devices [3.2.31] [3.2.25]

Returns the number of target devices.

```
integer function omp_get_num_devices()
```

omp_get_num_teams [3.2.32] [3.2.26]

Returns the number of teams in the current **teams** region, or 1 if called from outside of a **teams** region.

```
integer function omp_get_num_teams()
```

omp_get_team_num [3.2.33] [3.2.27]

Returns the team number of the calling thread. The team number is an integer between 0 and one less than the value returned by **omp_get_num_teams**, inclusive.

```
integer function omp_get_team_num()
```

omp_is_initial_device [3.2.34] [3.2.28]

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

```
integer function omp_is_initial_device()
```

omp_get_initial_device [3.2.35]

Returns a device number representing the host device.

```
integer function omp_get_initial_device()
```

omp_get_max_task_priority [3.2.36]

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

```
integer function omp_get_max_task_priority()
```

Lock Routines

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

Initialize lock [3.3.1] [3.3.1]

Initialize an OpenMP lock.

```
subroutine omp_init_lock(svar)
integer (kind=omp_lock_kind) svar

subroutine omp_init_nest_lock(nvar)
integer (kind=omp_nest_lock_kind) nvar
```

Initialize lock with hint [3.3.2]

Initialize an OpenMP lock with a hint.

```
subroutine omp_init_lock_with_hint(svar, hint)
integer (kind=omp_lock_kind) svar
integer (kind=omp_lock_hint_kind) hint

subroutine omp_init_nest_lock_with_hint(nvar, hint)
integer (kind=omp_nest_lock_kind) nvar
integer (kind=omp_lock_hint_kind) hint
```

omp_nest_lock_hint_kind:

omp_lock_hint_none	= 0
omp_lock_hint_uncontended	= 1
omp_lock_hint_contended	= 2
omp_lock_hint_nonspeculative	= 4
omp_lock_hint_speculative	= 8

Destroy lock [3.3.3] [3.3.2]

Ensure that the OpenMP lock is uninitialized.

```
subroutine omp_destroy_lock(svar)
integer (kind=omp_lock_kind) svar

subroutine omp_destroy_nest_lock(nvar)
integer (kind=omp_nest_lock_kind) nvar
```

Set lock [3.3.4] [3.3.3]

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

```
subroutine omp_set_lock(svar)
integer (kind=omp_lock_kind) svar

subroutine omp_set_nest_lock(nvar)
integer (kind=omp_nest_lock_kind) nvar
```

Unset lock [3.3.5] [3.3.4]

Unsets an OpenMP lock.

```
subroutine omp_unset_lock(svar)
integer (kind=omp_lock_kind) svar

subroutine omp_unset_nest_lock(nvar)
integer (kind=omp_nest_lock_kind) nvar
```

Test lock [3.3.6] [3.3.5]

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

```
logical function omp_test_lock(svar)
integer (kind=omp_lock_kind) svar

integer function omp_test_nest_lock(nvar)
integer (kind=omp_nest_lock_kind) nvar
```

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 [3.4.1] [3.4.1]

Returns elapsed wall clock time in seconds.

```
double precision function omp_get_wtime()
```

omp_get_wtick [3.4.2] [3.4.2]

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

```
double precision function omp_get_wtick()
```

Clauses

The set of clauses that is valid on a particular directive is described with the directive. Most clauses accept a comma-separated list of list items. All list items appearing in a clause must be visible, according to the scoping rules of the base language. Not all of the clauses listed in this section are valid on all directives.

If Clause [2.12]

The effect of the **if** clause depends on the construct to which it is applied.

if(*[directive-name-modifier :] scalar-logical-expression)*)

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.

Depend Clause [2.13.9]

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

depend(*dependence-type* **:** *list*)

Where *dependence-type* may be **in**, **out**, or **inout**:

> **in**: The generated task will be a dependent task of all previously generated sibling tasks that reference at least one of the list items in an **out** or **inout** *dependence-type* list.
>
> **out** and **inout**: The generated task will be a dependent task of all previously generated sibling tasks that reference at least one of the list items in an **in**, **out**, or **inout** *dependence-type list*.

depend(*dependence-type*)

Where *dependence-type* may be **source**.

depend(*dependence-type [*: *vec]*)

Where *dependence-type* may be **sink** and is the iteration vector, which has the form:

x_1 [± d_1], x_2 [± d_2], . . . , x_n [± d_n]

Data Sharing Attribute Clauses [2.15.3] [2.9.3]

Data-sharing attribute clauses apply only to variables whose names are visible in the construct on which the clause appears.

default(private | firstprivate |shared | none)

Explicitly determines the default data-sharing attributes of variables that are referenced in a **parallel**, **teams**, or task generating construct, causing all variables referenced in the construct that have implicitly determined data-sharing attributes to be shared.

shared(*list*)

Declares one or more list items to be shared by tasks generated by a **parallel**, **teams**, or task generating construct. The programmer must ensure that storage shared by an explicit **task** region does not reach the end of its lifetime before the explicit task region completes its execution.

private(*list*)

Declares one or more list items to be private to a task or a SIMD lane. Each task that references a list item that appears in a **private** clause in any statement in the construct receives a new list item.

firstprivate(*list*)

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

lastprivate(*list*)

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

linear(*linear-list[:linear-step]*)

Declares one or more list items to be private to a SIMD lane and to have a linear relationship with respect to the iteration space of a loop. Clause *linear-list* is *list* or *modifer*(*list*). *modifier* may be one of **ref**, **val**, or **uval**.

reduction(*reduction-identifier* **:** *list*)

Specifies a *reduction-identifier* and one or more list items. The *reduction-identifier* must match a previously declared *reduction-identifier* of the same name and type for each of the list items.

Implicitly Declared Fortran reduction-identifiers

Identifier	Initializer	Combiner
+	omp_priv = 0	omp_out = omp_in + omp_out
*****	omp_priv = 1	omp_out = omp_in * omp_out
-	omp_priv = 0	omp_out = omp_in + omp_out
.and.	omp_priv = .true.	omp_out = omp_in .and. omp_out
.or.	omp_priv = .false.	omp_out = omp_in .or. omp_out
.eqv.	omp_priv = .true.	omp_out = omp_in .eqv. omp_out
.neqv.	omp_priv = .false.	omp_out = omp_in .neqv. omp_out
max	omp_priv = *Least representable number in the reduction list item type*	omp_out = max(omp_in, omp_out)
min	omp_priv = *Largest representable number in the reduction list item type*	omp_out = min(omp_in, omp_out)
iand	omp_priv = *All bits on*	omp_out = iand(omp_in, omp_out)
ior	omp_priv = 0	omp_out = ior(omp_in, omp_out)
ieor	omp_priv = 0	omp_out = ieor(omp_in, omp_out)

SIMD Clauses [2.8.1]

safelen(*length*)

If used then no two iterations executed concurrently with SIMD instructions can have a greater distance in the logical iteration space than its value.

collapse(*n*)

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

simdlen(*length*)

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

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

Declares one or more list items to be aligned to the specified number of bytes. *alignment*, if present, must be a constant positive integer expression. If no optional parameter is specified, implementation-defined default alignments for SIMD instructions on the target platforms are assumed.

uniform(*argument-list*)

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

inbranch

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

notinbranch

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

Data Copying Clauses [2.14.4] [2.9.4]

copyin(*list*)

Copies the value of the master thread's threadprivate variable to the threadprivate variable of each other member of the team executing the **parallel** region.

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 Clause [2.14.5]

map(*[map-type*:*]list*)

Map a variable from the task's data environment to the device data environment associated with the construct.

map-type:

> **alloc:** On entry to the region each new corresponding list item has an undefined initial value.
>
> **to:** On entry to the region each new corresponding list item is initialized with the original list item's value.
>
> **from:** On exit from the region the corresponding list item's value is assigned to each original list item
>
> **tofrom:** (Default) On entry to the region each new corresponding list item is initialized with the original list item's value, and on exit from the region the corresponding list item's value is assigned to each original list item.
>
> **release:** On exit from the region, the corresponding list item's reference count is decremented by one.
>
> **delete:** On exit from the region, the corresponding list item's reference count is set to zero.

map-type-modifer:

> Must be **always**.

Defaultmap Clause [2.15.5.2]

defaultmap(tofrom:scalar)

Causes all scalar variables referenced in the construct that have implicitly determined data-mapping attributes to have the **tofrom** *map-type*.

Tasking Clauses [2.9]

final(*scalar-logical-expr*)

The generated task will be a final task if the final expression evaluates to true.

mergeable

Specifies that the generated task is a mergeable task.

priority(*priority-value*)

A non-negative numerical scalar expression that specifies a hint for the priority of the generated task.

grainsize(*grain-size*)

Causes the number of logical loop iterations assigned to each created task to be greater than or equal to the minimum of the value of the *grain-size* expression and the number of logical loop iterations, but less than two times the value of the *grain-size* expression.

num_tasks(*num-tasks*)

Create as many tasks as the minimum of the *num-tasks* expression and the number of logical loop iterations.

Environment Variables [4]

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

[4.11] [4.11] **OMP_CANCELLATION** *policy*
Sets the *cancel-var* ICV. *policy* may be **true** or **false**. If **true**, the effects of the cancel construct and of cancellation points are enabled and cancellation is activated

[4.13] [4.13] **OMP_DEFAULT_DEVICE** *device*
Sets the *default-device-var* ICV that controls the default device number to use in device constructs.

[4.12] [4.12] **OMP_DISPLAY_ENV** *var*
If *var* is **TRUE**, instructs the runtime to display the OpenMP version number and the value of the ICVs associated with the environment variables as *name=value* pairs. If *var* is **VERBOSE**, the runtime may also display vendor-specific variables. If *var* is **FALSE**, no information is displayed.

[4.3] [4.3] **OMP_DYNAMIC** *dynamic*
Sets the *dyn-var* ICV. If **true**, the implementation may dynamically adjust the number of threads to use for executing **parallel** regions.

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

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

[4.6] [4.6] **OMP_NESTED** *nested*
Sets the *nest-var* ICV to enable or to disable nested parallelism. Valid values for *nested* are **true** or **false**.

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

[4.5] [4.5] **OMP_PLACES** *places*
Sets the *place-partition-var* ICV that defines the OpenMP places available to the execution environment. *places* is an abstract name (**threads**, **cores**, **sockets**, or implementation-defined), or a list of non-negative numbers.

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

[4.1] [4.1] **OMP_SCHEDULE** *type[,chunk]*
Sets the *run-sched-var* ICV for the runtime schedule type and chunk size. Valid OpenMP schedule types are **static**, **dynamic**, **guided**, or **auto**.

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

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

[4.8] [4.8] **OMP_WAIT_POLICY** *policy*
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**.

ICV Environment Variable Values

The host and target device ICVs are initialized before any OpenMP API construct or OpenMP API routine executes. After the initial values are assigned, the values of any OpenMP environment variables that were set by the user are read and the associated ICVs for the host device are modified accordingly. The method for initializing a target device's ICVs is implementation defined.

Table of ICV Initial Values (Table 2.1) and Ways to Modify and to Retrieve ICV Values (Table 2.2) [2.3.2-3] [2.3.2-3]

ICV	Environment variable	Initial value	Ways to modify value	Ways to retrieve value	Ref.
dyn-var	**OMP_DYNAMIC**	Initial value is implementation defined if the implementation supports dynamic adjustment of the number of threads; otherwise, the initial value is *false*.	**omp_set_dynamic()**	**omp_get_dynamic()**	Sec 4.3
nest-var	**OMP_NESTED**	*false*	**omp_set_nested()**	**omp_get_nested()**	Sec 4.6
nthreads-var	**OMP_NUM_THREADS**	Implementation defined. The value of this ICV is a list.	**omp_set_num_threads()**	**omp_get_max_threads()**	Sec 4.2
run-sched-var	**OMP_SCHEDULE**	Implementation defined	**omp_set_schedule()**	**omp_get_schedule()**	Sec 4.1
def-sched-var	(none)	Implementation defined	(none)	(none)	---
bind-var	**OMP_PROC_BIND**	Implementation defined. The value of this ICV is a list.	(none)	**omp_get_proc_bind()**	Sec 4.4
stacksize-var	**OMP_STACKSIZE**	Implementation defined	(none)	(none)	Sec 4.7
wait-policy-var	**OMP_WAIT_POLICY**	Implementation defined	(none)	(none)	Sec 4.8
thread-limit-var	**OMP_THREAD_LIMIT**	Implementation defined	**thread_limit** *clause*	**omp_get_thread_limit()**	Sec 4.10
max-active-levels-var	**OMP_MAX_ACTIVE_LEVELS**	The initial value is the number of levels of parallelism that the implementation supports.	**omp_set_max_active_levels()**	**omp_get_max_active_levels()**	Sec 4.9
active-levels-var	(none)	*zero*	(none)	**omp_get_active_level()**	---
levels-var	(none)	*zero*	(none)	**omp_get_level()**	---

place-partition-var	**OMP_PLACES**	Implementation defined	(none)	**omp_get_partition_num_places()**
				omp_get_partition_place_nums()
				omp_get_place_num_procs()
				omp_get_place_proc_ids()
					Sec 4.5

cancel-var	**OMP_CANCELLATION**	*false*	(none)	**omp_get_cancellation()**	Sec 4.11
default-device-var	**OMP_DEFAULT_DEVICE**	Implementation defined	**omp_set_default_device()**	**omp_get_default_device()**	Sec 4.13
max-task-priority-var	**OMP_MAX_TASK_PRIORITY**	*zero*	(none)	**omp_get_max_task_priority()**	Sec 4.14

Notes