### 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 directives. 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 directives.

**parallel** [2.5] [2.5]

Forms a team of threads and starts parallel execution.

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

**structured-block**

```fortran
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] [2.5.1]

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

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

**private(list)**

```fortran
private(list)
```

**firstprivate(list)**

```fortran
firstprivate(list)
```

**lastprivate(list)**

```fortran
lastprivate(list)
```

```fortran
linear(list) [: linear-step])
reduction(reduction-identifier : list)
schedule(modifier [, modifier] : ] kind[, chunk_size])
collapse(n)
ordered [(i|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** [2.8.1] [2.8.1]

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

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

**structured-block**

```fortran
structured-block
```

### 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.

```fortran
!$omp sections [clause[ [, clause] ...] ]
```

**structured-block**

```fortran
structured-block
```

```fortran
... 
```

```fortran
!$omp end sections [nowait] 
```

**clause:**

```fortran
private(list)
firstprivate(list)
lastprivate(list)
reduction(reduction-identifier : list)
```

**nowait**

```fortran
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.

```fortran
!$omp workshare 
```

**structured-block**

```fortran
structured-block
```

```fortran
!$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.

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

**clause:**

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

**Nowait**

```fortran
nowait
```

**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.

```fortran
!$omp task [clause[ [, clause] ...] ]
```

**structured-block**

```fortran
structured-block
```

```fortran
!$omp end task 
```

**clause may be:**

```fortran
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)
```
## 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.

```fortran
!$omp target [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)
- gransiz[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.

```fortran
!$omp target 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.

```fortran
!$omp taskyield
```

### target data [2.10.1] [2.9.1]
Creates a device data environment for the extent of the region.

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

|$omp target end 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.

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

|$omp target end data]
```

clause:

- if( target enter data : ) scalar-logical-expression
- device(scalar-integer-expression)
- map([map-type-modifier[,]] map-type : ] list)
- dep[endency-type : ] list
- nowait

### target exit data [2.10.3]
Specifies that list items are unmapped from a device data environment.

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

|$omp target end data]
```

clause:

- if( target exit data : ) scalar-logical-expression
- device(scalar-integer-expression)
- map([map-type-modifier[,]] map-type : ] list)
- dep[endency-type : ] list
- nowait

### 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.

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

|$omp target end update]
```

clause:

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

### declare target [2.10.6] [2.9.4]
A declarative directive that specifies that variables and functions are mapped to a device.

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

|$omp declare target]
```

clause:

- to(list)
- from(list)

### distribute simd [2.10.9] [2.9.7]
Specifies loops which are executed concurrently using SIMD instructions.

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

|$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.

```fortran
!$omp distribute parallel do [clause [ , ]clause] ...

|$omp end distribute parallel do]
```

clause:

- Any accepted by the distribute or parallel do 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.

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

|$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.

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

|$omp end parallel sections]
```

clause:

- Any of the clauses accepted by the parallel or sections directives, with identical meanings and restrictions.

### parallel workshare [2.11.3] [2.10.3]
Shortcut for specifying a parallel construct containing one workshare construct and no other statements.

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

|$omp end parallel workshare]
```

clause:

- Any of the clauses accepted by the parallel directive, with identical meanings and restrictions.
Directives and Constructs for Fortran (continued)

**parallel do** [2.11.4] [2.10.4]
Shortcut for specifying a parallel construct containing one do simd construct and no other statements.

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

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

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

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

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

|$omp target teams [clause [, clause] ...]
|$omp end target teams
class: 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.

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

|$omp teams distribute simd [clause [, clause] ...]
|$omp end teams distribute simd
clause: Any accepted by the teams or distribute 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] ...]
|$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.

|$omp critical [name] [hint(hint-expression)]
|$omp end critical

**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:

|$omp atomic [seq_cst] [, read [, seq_cst]]
|$omp atomic [seq_cst] [, write [, seq_cst]]
|$omp atomic [seq_cst] [, update [, seq_cst]]
|$omp atomic [seq_cst] [, capture [, seq_cst]]
|$omp atomic [seq_cst] [, destroy [, seq_cst]]
|$omp atomic [seq_cst] [, release [, seq_cst]]
|$omp atomic [seq_cst] [, detach [, seq_cst]]

(atomic continues on the next page)
## Directives and Constructs for Fortran (continued)

### atomic (continued)

```fortran
!$omp atomic [seq_cst,] capture [,...] capture-statement
update-statement
!$omp end atomic
```

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

- capture-statement
- write-statement
- x = x
- x = expr
- 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, IEQV, .OR., NEQV.

### ordered (continued)

```fortran
!$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.

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

#### 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.

```fortran
!$omp cancellation point construct-type-clause
construct-type-clause:
  parallel sections do taskgroup
if-clause:
  if(scalar-logical-expression)
```

### 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 specified.

```fortran
!$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.

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

- clause:
  - threads
  - simd

(ordered continues in the next column)

### Runtime Library Routines for Fortran

**Execution Environment Routines**

- **omp_set_num_threads** [3.2.10] [3.2.11]
  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.

```fortran
subroutine omp_set_num_threads(num_threads)
integer num_threads
description
```

- **omp_get_num_threads** [3.2.2] [3.2.2]
  Returns the number of threads used in the current task. 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.

```fortran
description
```

- **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.

```fortran
description
```

- **omp_get_thread_num** [3.2.4] [3.2.4]
  Returns the number of the calling thread, within the current team.

```fortran
description
```

- **omp_get_max_threads** [3.2.3] [3.2.3]
  Returns the number of threads in the current team.

```fortran
description
```

- **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.

```fortran
description
```

- **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.

```fortran
description
```

- **omp_get_schedule** [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.

```fortran
description
```

- **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.

```fortran
description
```

- **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.

```fortran
description
```

- **omp_set_nested** [3.2.10] [3.2.10]
  Enables or disables nested parallelism, by setting the nest-var ICV.

```fortran
description
```

- **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.

```fortran
description
```

- **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.

```fortran
description
```

- **omp_get_schedule** [3.2.12] [3.2.12]
  This routine returns the value of the sched-var ICV.

```fortran
description
```

- **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.

```fortran
description
```

- **omp_get_schedule** [3.2.12] [3.2.12]
  This routine returns the value of the sched-var ICV.

```fortran
description
```

- **omp_get_schedule** [3.2.12] [3.2.12]
  This routine returns the value of the sched-var ICV.

```fortran
description
```
Runtime Library Routines for Fortran (continued)

`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.

`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.

`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.

`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.

`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.

`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.

`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.

`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.

`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.

`omp_get_team_num` [3.2.22] [3.2.22]
Returns the number of teams in the current execution environment in the specified place.

`omp_get_num_places` [3.2.23] [3.2.23]
Returns the number of places available to the execution environment in the place list.

`omp_get_place_num` [3.2.24] [3.2.24]
Returns the number of processors available to the execution environment in the specified place.

`omp_get_max_task_priority` [3.2.25] [3.2.25]
Returns numerical identifiers of the processors available to the execution environment in the specified place.

`omp_get_num_devices` [3.2.26] [3.2.26]
Returns the place number of the place to which the encountering thread is bound.

`omp_get_partition_num_places` [3.2.27] [3.2.27]
Returns the number of places in the place partition of the current execution environment in the specified place.

`omp_get_partition_num_places` [3.2.28] [3.2.28]
Returns the list of place numbers corresponding to the places in the place-partition-var ICV of the innermost implicit task.

`omp_get_num_devices` [3.2.29] [3.2.29]
Assigns the value of the default-device-var ICV, which determines default target device.

`omp_get_default_device` [3.2.30] [3.2.30]
Returns the value of the default-device-var ICV, which determines default target device.

`omp_get_num_devices` [3.2.31] [3.2.31]
Returns the number of target devices.

`omp_get_num_teams` [3.2.32] [3.2.32]
Returns the number of teams in the current teams region, or 1 if called from outside of a teams region.

`omp_get_num_teams` [3.2.32] [3.2.32]
Returns the number of teams in the current teams region, or 1 if called from outside of a teams region.

`omp_get_team_num` [3.2.33] [3.2.33]
Returns the number of teams in the calling thread. The team number is an integer between 0 and one less than the value returned by `omp_get_num_teams`, inclusive.

`omp_is_initial_device` [3.2.34] [3.2.34]
Returns true if the current task is executing on the host device; otherwise, it returns false.

`omp_get_max_task_priority` [3.2.35] [3.2.35]
Returns a device number representing the host device.

`omp_get_max_task_priority` [3.2.36] [3.2.36]
Returns the maximum value that can be specified in the priority clause.

`omp_get_proc_bind` [3.2.37] [3.2.37]
Returns the thread affinity policy to be used for the subsequent nested parallel regions that do not specify a proc_bind clause.

`omp_get_place_proc_ids` [3.2.38] [3.2.38]
Returns the thread affinity policy to be used for the subsequent nested parallel regions that do not specify a proc_bind clause.

`omp_get_initial_device` [3.2.39] [3.2.39]
Returns true if the current task is executing on the host device; otherwise, it returns false.

`omp_get_initial_device` [3.2.39] [3.2.39]
Returns a device number representing the host device.

`omp_get_initial_device` [3.2.39] [3.2.39]
Returns the maximum value that can be specified in the priority clause.

`omp_get_num_places` [3.2.40] [3.2.40]
Returns the number of places available to the execution environment in the place list.

`omp_get_place_proc_ids` [3.2.41] [3.2.41]
Returns the place number of the place to which the encountering thread is bound.

`omp_get_initial_device` [3.2.42] [3.2.42]
Returns the number of processors available to the execution environment in the specified place.

`omp_get_max_task_priority` [3.2.43] [3.2.43]
Returns numerical identifiers of the processors available to the execution environment in the specified place.

`omp_get_current_device` [3.2.44] [3.2.44]
Returns the value of the default-device-var ICV, which determines default target device.

`omp_set_schedule` [3.2.45] [3.2.45]
Sets the value of run-sched-var ICV.

`omp_set_schedule` [3.2.46] [3.2.46]
Sets the value of run-sched-var ICV.

`omp_set_schedule` [3.2.47] [3.2.47]
Sets the value of run-sched-var ICV.

`omp_set_free` [3.2.48] [3.2.48]
Unsets the current task region.

`omp_set_initial_device` [3.2.49] [3.2.49]
Sets the thread affinity policy to be used for the subsequent nested parallel regions that do not specify a proc_bind clause.

`omp_set_proc_bind` [3.2.50] [3.2.50]
Sets the thread affinity policy to be used for the subsequent nested parallel regions that do not specify a proc_bind clause.

`omp_set_max_active_levels` [3.2.51] [3.2.51]
Limits the number of nested active parallel regions, by setting max-active-levels-var ICV.

`omp_set_max_active_levels` [3.2.52] [3.2.52]
Limits the number of nested active parallel regions, by setting max-active-levels-var ICV.

`omp_set_schedule` [3.2.53] [3.2.53]
Sets the value of run-sched-var ICV.

`omp_set_schedule` [3.2.54] [3.2.54]
Sets the value of run-sched-var ICV.

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.

`initiate lock` [3.3.1] [3.3.1]
Initialize an OpenMP lock.

`openmp_init_lock` [3.3.2] [3.3.2]
Initialize an OpenMP lock with a hint.

`openmp_init_nest_lock` [3.3.2] [3.3.2]
Initialize an OpenMP lock with a hint.

`reset lock` [3.3.3] [3.3.3]
Reset an OpenMP lock.

`reset lock` [3.3.4] [3.3.4]
Reset an OpenMP lock.

`reset lock` [3.3.5] [3.3.5]
Reset an OpenMP lock.

Timing Routines
Timing routines support a portable wall clock timer. These routines 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.

`omp_get_wtick` [3.4.2] [3.4.2]
Returns the precision of the timer (seconds between ticks) used by `omp_get_wtime`.

`omp_set_max_thread_limit` [3.4.3] [3.4.3]
Sets the maximum number of threads allowed in the current execution environment.

`omp_set_max_thread_limit` [3.4.4] [3.4.4]
Sets the maximum number of threads allowed in the current execution environment.

`omp_set_max_thread_limit` [3.4.5] [3.4.5]
Sets the maximum number of threads allowed in the current execution environment.
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 in or inout dependence-type list.
out: 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.

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.

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:
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.
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]

<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>Ref.</th>
</tr>
</thead>
<tbody>
<tr>
<td>dyn-var</td>
<td>OMP_DYNAMIC</td>
<td>Initial value is 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>Sec 4.3</td>
</tr>
<tr>
<td>nest-var</td>
<td>OMP_NESTED</td>
<td>false</td>
<td>omp_set_nested()</td>
<td>omp_get_nested()</td>
<td>Sec 4.6</td>
</tr>
<tr>
<td>nthreads-var</td>
<td>OMP_NUM_THREADS</td>
<td>Implementation defined. The value of this ICV is a list.</td>
<td>omp_set_num_threads()</td>
<td>omp_get_max_threads()</td>
<td>Sec 4.2</td>
</tr>
<tr>
<td>run-sched-var</td>
<td>OMP_SCHEDULE</td>
<td>Implementation defined</td>
<td>omp_set_schedule()</td>
<td>omp_get_schedule()</td>
<td>Sec 4.1</td>
</tr>
<tr>
<td>def-sched-var</td>
<td>(none)</td>
<td>Implementation defined</td>
<td>(none)</td>
<td>(none)</td>
<td>---</td>
</tr>
<tr>
<td>bind-var</td>
<td>OMP_PROC_BIND</td>
<td>Implementation defined. The value of this ICV is a list.</td>
<td>(none)</td>
<td>omp_get_proc_bind()</td>
<td>Sec 4.4</td>
</tr>
<tr>
<td>stacksize-var</td>
<td>OMP_STACKSIZE</td>
<td>Implementation defined</td>
<td>(none)</td>
<td>(none)</td>
<td>Sec 4.7</td>
</tr>
<tr>
<td>wait-policy-var</td>
<td>OMP_WAIT_POLICY</td>
<td>Implementation defined</td>
<td>(none)</td>
<td>(none)</td>
<td>Sec 4.8</td>
</tr>
<tr>
<td>thread-limit-var</td>
<td>OMP_THREAD_LIMIT</td>
<td>Implementation defined</td>
<td>thread_limit clause</td>
<td>omp_get_thread_limit()</td>
<td>Sec 4.10</td>
</tr>
<tr>
<td>max-active-levels-var</td>
<td>OMP_MAX_ACTIVE_LEVELS</td>
<td>The initial value is the number of levels of parallelism that the implementation supports.</td>
<td>omp_set_max_active_levels()</td>
<td>omp_get_max_active_levels()</td>
<td>Sec 4.9</td>
</tr>
<tr>
<td>active-levels-var</td>
<td>(none)</td>
<td>zero</td>
<td>(none)</td>
<td>omp_get_active_level()</td>
<td>---</td>
</tr>
<tr>
<td>levels-var</td>
<td>(none)</td>
<td>zero</td>
<td>(none)</td>
<td>omp_get_level()</td>
<td>---</td>
</tr>
<tr>
<td>place-partition-var</td>
<td>OMP_PLACES</td>
<td>Implementation defined</td>
<td>(none)</td>
<td>omp_get_partition_num_places()</td>
<td>Sec 4.5</td>
</tr>
<tr>
<td>cancel-var</td>
<td>OMP_CANCELLATION</td>
<td>false</td>
<td>(none)</td>
<td>omp_get_cancellation()</td>
<td>Sec 4.11</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>Sec 4.13</td>
</tr>
<tr>
<td>max-task-priority-var</td>
<td>OMP_MAX_TASK_PRIORITY</td>
<td>zero</td>
<td>(none)</td>
<td>omp_get_max_task_priority()</td>
<td>Sec 4.14</td>
</tr>
</tbody>
</table>