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 clauses.

```
purpose
```

Forms a team of threads and starts parallel execution.

```
(purpose-clause
  [end-purpose-clause]
) ...
```

Forms a team of threads and starts parallel execution.

```
(distribute-clause
  [end-distribute-clause]
) ...
```

Forms a team of threads and starts parallel execution.

```
(workshare-clause
  [end-workshare-clause]
) ...
```

Forms a team of threads and starts parallel execution.

```
(declare-target-clause
  [end-declare-target-clause]
) ...
```

Forms a team of threads and starts parallel execution.

```
(target-clause
  [end-target-clause]
) ...
```

Forms a team of threads and starts parallel execution.

```
(distribute-simd-clause
  [end-distribute-simd-clause]
) ...
```

Forms a team of threads and starts parallel execution.

```
(declare-simd-clause
  [end-declare-simd-clause]
) ...
```

Forms a team of threads and starts parallel execution.
Directives (Continued)

65 distribute parallel do [simd] [2.9.8, 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.

|$omp distribute parallel do [clause [, clause ...]]
do-loops
|$omp end distribute parallel do/
clause: Any accepted by the distribute or parallel loop [SIMD] directives.

parallel do [2.10.1] [2.6.1]
Shortcut for specifying a parallel construct containing one or more associated loops and no other statements.

|$omp parallel do [clause [, clause ...]]
do-loop
|$omp end parallel do/
clause: Any accepted by the parallel or do directives.

parallel sections [2.10.2] [2.6.2]
Shortcut for specifying a parallel construct containing one or more associated loops and no other statements.

|$omp parallel sections [clause [, clause ...]]
|$omp section
structured-block
|$omp end section
structured-block
 ...
|$omp end parallel sections
clause: Any of the clauses accepted by the parallel or sections directives.

parallel workshare [2.10.3] [2.6.3]
Shortcut for specifying a parallel construct containing one workshare construct and no other statements.

|$omp parallel workshare [clause [, clause ...]]
|$omp section
structured-block
|$omp end parallel workshare
clause: Any of the clauses accepted by the parallel directive, with identical meanings and restrictions.

parallel do simd [2.10.4]
Shortcut for specifying a parallel construct containing one loop 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, do or simd directives with identical meanings and restrictions.

69 target teams [2.10.5]
Shortcut for specifying a target construct containing a teams construct.

|$omp target teams [clause [, clause ...]]
structured-block
|$omp end target teams
clause: See clause for target or teams

69 target teams distribute [simd] [2.10.6, 2.10.7]
Shortcuts for specifying target constructs containing a distribute or distribute [simd] construct.

|$omp target teams distribute [simd] [clause [, clause ...]]
do-loops
|$omp end target teams distribute [simd] /
clause: Any clause used for teams or distribute [simd]

69 target teams distribute [simd] [2.10.8, 2.10.9]
Shortcuts for specifying target constructs containing a teams distribute [simd] construct.

|$omp target teams distribute [simd] [clause [, clause ...]]
do-loops
|$omp end target teams distribute [simd] /
clause: Any clause used for target or teams distribute [simd]

69 target teams distribute parallel do [simd] [2.10.10, 12]
Shortcuts for specifying teams constructs containing a distribute parallel loop [simd] construct.

|$omp target teams distribute parallel do [simd] [clause [, clause ...]]
do-loops
|$omp end target teams distribute parallel do [simd] /
clause: Any clause used for teams or distribute parallel do [simd]

69 target teams distribute parallel do [simd] [2.10.11, 13]
Shortcuts for specifying target constructs containing a teams distribute parallel do [simd] construct.

|$omp target teams distribute parallel do [simd] &
{clause [, clause ...]}
do-loops
|$omp end target teams distribute parallel do [simd] /
clause: Any clause used for target or teams distribute parallel do [simd]

task [2.11.1] [2.7.3]
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(scalar-logical-expression)
final(scalar-logical-expression)
untied
default/private | firstprivate | shared | none
mergeable
private(list)
firstprivate(list)
shared(list)
|$omp depend(dependence-type : list)

69 taskwait [2.12.4] [2.8.4]
taskgroup [2.12.5]
These constructs each specify a wait on the completion of child tasks of the current task. taskgroup also waits for descendant tasks.

|$omp taskwait
|$omp taskgroup
structured-block
|$omp end taskgroup

atomic [2.12.6] [2.8.5]
Ensures a specific storage location is accessed atomically.

|$omp atomic declare reduction [seq_cst]
capture-stmt
|$omp atomic declare reduction [seq_cst]
write-stmt
|$omp atomic declare reduction [seq_cst]
capture-stmt
|$omp atomic declare reduction [seq_cst]
write-stmt
|$omp atomic declare reduction [seq_cst]
capture-stmt
|$omp atomic declare reduction [seq_cst]
write-stmt

flush [2.12.7] [2.8.6]
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 [2.12.8] [2.8.7]
Specifies a structured block in a loop region that will be executed in the order of the loop iterations.

|$omp ordered structured-block
|$omp end ordered

9 cancel [2.13.1]
Requests cancellation of the innermost enclosing region of the type specified.

|$omp cancel construct-type-clause [ , ] [if-clause]
parallel sections do
taskgroup
if-clause:
if(scalar-logical-expression)

9 cancellation point [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 [2.14.2] [2.9.2]
Specifies that variables are replicated, with each thread having its own copy.

|$omp threadprivate([list]) list: A comma-separated list of named variables and named common blocks.

9 declare reduction [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, itor.
type-list: A list of type specifiers
combiner: An assignment statement or a subroutine name followed by an argument list
initializer-clause: initializer (omp_priv = expression-or-subroutine-name [argument-list])
Runtime Library Routines

Execution Environment Routines

\textit{omp_set_num_threads} [3.2.1] [3.2.1]

Affects the number of threads used for subsequent parallel regions not specifying a \texttt{nthreads} clause, by setting the value of the first element of the \texttt{nthreads} ICV of the current task to \texttt{nthreads}.

\texttt{subroutine omp_set_num_threads(num_threads)}
\texttt{integer num_threads}

\textit{omp_get_num_threads} [3.2.2] [3.2.2]

Returns the number of threads in the current team. The binding region for an \textit{omp_get_num_threads} region is the innermost enclosing parallel region. If called from the sequential part of a program, this routine returns 1.

\texttt{integer function omp_get_num_threads()}

\textit{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 \texttt{nthreads} clause were encountered after execution returns from this routine.

\texttt{integer function omp_get_max_threads()}

\textit{omp_get_thread_num} [3.2.4] [3.2.4]

Returns the thread number of the calling thread, within the team executing the parallel region.

\texttt{integer function omp_get_thread_num()}

\textit{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.

\texttt{integer function omp_get_num_procs()}

\textit{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.

\texttt{logical function omp_in_parallel()}

\textit{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.

\texttt{subroutine omp_set_dynamic(dynamic_threads)}
\texttt{logical dynamic_threads}

\textit{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.

\texttt{logical function omp_get_dynamic()}

\textit{omp_get_cancellation} [3.2.9]

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

\texttt{logical function omp_get_cancellation()}

\textit{omp_set_nested} [3.2.10] [3.2.9]

Enables or disables nested parallelism, by setting the \texttt{nested} variate.

\texttt{subroutine omp_set_nested(nested)}
\texttt{logical nested}

\textit{omp_get_nested} [3.2.11] [3.2.10]

Returns the value of the \texttt{nested} variate, which indicates if nested parallelism is enabled or disabled.

\texttt{logical function omp_get_nested()}

\textit{omp_set_schedule} [3.2.12] [3.2.11]

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

\texttt{subroutine omp_set_schedule(kind, modifier)}
\texttt{integer (kind=omp_sched_kind) kind}
\texttt{integer modifier}

See kind for \textit{omp_get_schedule}.

\textit{omp_get_schedule} [3.2.13] [3.2.12]

Returns the value of \texttt{run-sched-var} ICV, which is the schedule applied when runtime schedule is used.

\texttt{subroutine omp_get_schedule(kind, modifier)}
\texttt{integer (kind=omp_sched_kind) kind}
\texttt{integer modifier}

\textit{omp_set_schedule} and \textit{omp_get_schedule} is an implementation-defined schedule or:
\texttt{omp_sched_static = 1}
\texttt{omp_sched_dynamic = 2}
\texttt{omp_sched_guided = 3}
\texttt{omp_sched_auto = 4}

\textit{omp_get_thread_limit} [3.2.14] [3.2.13]

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

\texttt{integer function omp_get_thread_limit()}

\textit{omp_set_max_active_levels} [3.2.15] [3.2.14]

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

\texttt{subroutine omp_set_max_active_levels(max_levels)}
\texttt{integer max_levels}

\textit{omp_get_max_active_levels} [3.2.16] [3.2.15]

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

\texttt{integer function omp_get_max_active_levels()}

\textit{omp_get_max_team_size} [3.2.17] [3.2.16]

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.

\texttt{integer function omp_get_max_team_size()}

\textit{omp_get_team_size} [3.2.18] [3.2.17]

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

\texttt{integer function omp_get_team_size(level) }
\texttt{integer level}

\textit{omp_get_active_level} [3.2.19] [3.2.18]

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.

\texttt{integer function omp_get_active_level()}

\textit{omp_set_active_level} [3.2.20] [3.2.19]

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.

\texttt{integer function omp_set_active_level()}
\texttt{integer level}

\textit{omp_get_team_num} [3.2.21] [3.2.20]

Returns the number of teams in the current teams region. If called from outside of a teams region.

\texttt{integer function omp_get_team_num()}

\textit{omp_get_num_devices} [3.2.22] [3.2.21]

Returns the number of target devices.

\texttt{integer function omp_get_num_devices()}

\textit{omp_get_num_teams} [3.2.23] [3.2.22]

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

\texttt{integer function omp_get_num_teams()}

\textit{omp_get_team_num} [3.2.24] [3.2.23]

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

\texttt{integer function omp_get_team_num()}

\textit{omp_is_initial_device} [3.2.25] [3.2.24]

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

\texttt{logical function omp_is_initial_device()}

Lock Routines

General-purpose lock routines.

\textit{Initialize lock} [3.3.1] [3.3.1]

Initializes an OpenMP lock.

\texttt{subroutine omp_init_lock(ivar)}
\texttt{integer (kind=omp_lock_kind) ivar}

\texttt{subroutine omp_init_nest_lock(nvar, svar)}
\texttt{integer (kind=omp_nest_lock_kind) nvar}
\texttt{logical (kind=omp_nest_lock_kind) svar}

\textit{Destroy lock} [3.3.2] [3.3.2]

Ensures that the OpenMP lock is uninitialized.

\texttt{subroutine omp_destroy_lock(ivar)}
\texttt{integer (kind=omp_lock_kind) ivar}

\texttt{subroutine omp_destroy_nest_lock(nvar, svar)}
\texttt{integer (kind=omp_nest_lock_kind) nvar}
\texttt{logical (kind=omp_nest_lock_kind) svar}

\textit{Set lock} [3.3.3] [3.3.3]

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

\texttt{subroutine omp_set_lock(ivar)}
\texttt{integer (kind=omp_lock_kind) ivar}

\texttt{subroutine omp_set_nest_lock(nvar, svar)}
\texttt{integer (kind=omp_nest_lock_kind) nvar}
\texttt{logical (kind=omp_nest_lock_kind) svar}

\textit{Unset lock} [3.3.4] [3.3.4]

Unsets an OpenMP lock.

\texttt{subroutine omp_unset_lock(ivar)}
\texttt{integer (kind=omp_lock_kind) ivar}

\texttt{subroutine omp_unset_nest_lock(nvar, svar)}
\texttt{integer (kind=omp_nest_lock_kind) nvar}
\texttt{logical (kind=omp_nest_lock_kind) svar}

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.

\textit{omp_get_wtick} [3.4.1] [3.4.1]

Returns elapsed wall clock time in seconds.

\texttt{double precision function omp_get_wtick()}

\textit{omp_get_wtime} [3.4.2] [3.4.1]

Returns the precision of the timer (seconds between ticks) used by \textit{omp_get_wtime}.

\texttt{double precision function omp_get_wtime()}
Environment Variables [4]

[4.9] 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.

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

[4.8] 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 VERBOS, the runtime may also display vendor-specific variables. If var is FALSE, no information is displayed.

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

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

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

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

[4.5] OMP_PACES 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] OMP_PROC_BIND policy
Sets the value of the 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] 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] 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] OMP_THREAD_LIMIT limit
Sets the thread-limit-var ICV that controls the number of threads participating in the OpenMP program.

[4.8] OMP_WAIT_POLICY policy
Sets the wait-policy-var ICV that provides a hint about the desired behavior of waiting threads. Valid values for policy are ACTIVE (waiting threads processor cycles while waiting) and PASSIVE.

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. The set of clauses that is valid on a particular directive is described with the directive.

Data Sharing Attribute Clauses [2.14.3] [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, task, or teams construct. The program 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.

shared(list)
Declares one or more list items to be shared by tasks generated by a parallel, task, or teams construct. The program 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(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.

Data Copying Clauses [2.14.4] [2.9.4]
copy(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)
Maps a variable from the task’s data environment to the device data environment associated with the construct.

Map-type:
alloc: On entry to the regions 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.

(Continued ->)

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.

Operators for reduction (initialization values):
+ [0] .eqv. (true.)
* [1] .neqv. (false.)
- [0] iand (All bits on)
.iand. (true.) ior (0)
.or. (false.) ior (0)
max (Least representable number in reduction list item type)
min (Largest representable number in reduction list item type)

SIMD Clauses [2.8.1, 2.8.2]
safe(len)
If used then no two iterations executed concurrently with SIMD instructions can have a greater distance in the logical iteration space than the value.

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

simd(len)
A constant positive integer expression that specifies the number of concurrent arguments of the function.

aligned(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, the default alignment that SIMD instructions in the target platforms use is assumed.

uniform(1st,alignment)
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.