OpenMP Fortran

OpenMP 4.0 API Fortran Syntax Quick Reference Card

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.

4.0 Refers to functionality new in version 4.0.

[n.n.n] refers to sections in the OpenMP API specification version 4.0, and [n.n.n] refers to version 3.1.

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 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 may not appear in **PURE** or **ELEMENTAL** procedures.

parallel [2.5] [2.4]

Forms a team of threads and starts parallel execution.

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

clause:
if(*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*)
4.0 **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*)
reduction(*reduction-identifier* **:** *list*)
schedule(*kind***[,** *chunk_size*])
collapse(*n*)
ordered

kind:

- **static:** Iterations are divided into chunks of size *chunk_size.*
- **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.

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] [2.5.3]

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] [2.5.4]

The **workshare** construct 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
FORALL, **WHERE**, **atomic**, **critical**, or **parallel** constructs

4.0 simd [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*)
linear(*list[:linear-step]*)
aligned(*list[:alignment]*)
private(*list*)
lastprivate(*list*)
reduction(*reduction-identifier***:** *list*)
collapse(*n*)

4.0 declare simd [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(*argument-list[:constant-linear-step]*)
aligned(*argument-list[:alignment]*)
uniform(*argument-list*)
inbranch
notinbranch

4.0 do simd [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.

4.0 target [data] [2.9.1, 2.9.2]

These constructs create a device data environment for the extent of the region. **target** also starts execution on the device.

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

clause:
device(*scalar-integer-expression*)
map(*[map-type :] list*)
if(*scalar-logical-expression*)

4.0 target update [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] ...]*

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

clause is *motion-clause* or one of:
device(*scalar-integer-expression*)
if(*scalar-logical-expression*)

4.0 declare target [2.9.4]

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

For functions and subroutines:
!$omp declare target

For variables, functions and subroutines:
!$omp declare target (*list*)
list: A comma-separated list of named variables, procedure names, and named common blocks.

4.0 teams [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 (for the **teams** construct):
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*)

4.0 distribute [simd] [2.9.6, 2.9.7]

distribute specifies loops which are executed by the thread teams. **distribute simd** specifies loops which are executed concurrently using SIMD instructions.

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

clause (for **distribute***)*:
private(*list*)
firstprivate(*list*)
collapse(*n*)
dist_schedule(*kind[, chunk_size]*)

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

Continued

Directives (Continued)

4.0 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 **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.

parallel workshare [2.10.3] [2.6.3]

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

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

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

4.0 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.

4.0 target teams [2.10.5]

Shortcut for specifying a **target** construct containing a **teams** construct.

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

clause: See *clause* for **target** or **teams**

4.0 teams distribute [simd] [2.10.6, 2.10.7]

Shortcuts for specifying **teams** constructs containing a distribute or **distribute [simd]** construct.

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

clause: Any clause used for **teams** or **distribute [simd]**

4.0 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]**

4.0 teams distribute parallel do [simd] [2.10.10, 12]

Shortcuts for specifying **teams** constructs containing a **distribute parallel loop [simd]** construct.

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

clause: Any *clause* used for **teams** or **distribute parallel do [simd]**

4.0 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] [2.7.1]

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(*scalar-logical-expression*)
final(*scalar-logical-expression*)
untied
default(**private** | **firstprivate** | **shared** | **none**)
mergeable
private(*list*)
firstprivate(*list*)
shared(*list*)
4.0 **depend**(*dependence-type* : *list*)

The list items that appear in the **depend** clause may include array sections.

dependence-type:

- **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** clause *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** clause.

taskyield [2.11.2] [2.7.2]

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

!$omp taskyield

master [2.12.1] [2.8.1]

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

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

critical [2.12.2] [2.8.2]

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

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

barrier [2.12.3] [2.8.3]

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 [2.12.4] [2.8.4], 4.0 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. *[*seq_cst*]* is 4.0. May take one of the following forms:

!$omp atomic read *[***seq_cst***]* *capture-stmt* *[***!$omp end atomic***]*	**!$omp atomic write** *[***seq_cst***]* *write-stmt* *[***!$omp end atomic***]*
!$omp atomic capture*[***seq_cst***]* *update-stmt* *capture-stmt* **!$omp end atomic**	**!$omp atomic capture** *[***seq_cst***]* *capture-stmt* *update-stmt* **!$omp end atomic**
!$omp atomic *[***update***]**[***seq_cst***]* *update-stmt* *[***!$omp end atomic***]*	**!$omp atomic capture** *[***seq_cst***]* *capture-stmt* *write-stmt* 4.0 **!$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 [2.12.7] [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

4.0 cancel [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*)

4.0 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.

4.0 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, ieor.

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

Return types are shown in green.

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 team executing the **parallel** region.

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.

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()

4.0 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*.

logical function omp_get_cancellation()

omp_set_nested [3.2.10] [3.2.9]

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

subroutine omp_set_nested(*nested*)
logical *nested*

omp_get_nested [3.2.11] [3.2.10]

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] [3.2.11]

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*, *modifier*)
integer (**kind=omp_sched_kind**) *kind*
integer *modifier*

See *kind* for **omp_get_schedule**.

omp_get_schedule [3.2.13] [3.2.12]

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

subroutine omp_get_schedule(*kind*, *modifier*)
integer (**kind=omp_sched_kind**) *kind*
integer *modifier*

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] [3.2.13]

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] [3.2.14]

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] [3.2.15]

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] [3.2.16]

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.

integer function omp_get_level()

omp_get_ancestor_thread_num [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.

integer function omp_get_ancestor_thread_num(*level*)
integer *level*

omp_get_team_size [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.

integer function omp_get_team_size(*level*)
integer *level*

omp_get_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.

integer function omp_get_active_level()

omp_in_final [3.2.21] [3.2.20]

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

logical function omp_in_final()

4.0 omp_get_proc_bind [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

4.0 omp_set_default_device [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*

4.0 omp_get_default_device [3.2.24]

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

integer function omp_get_default_device()

4.0 omp_get_num_devices [3.2.25]

Returns the number of target devices.

integer function omp_get_num_devices()

4.0 omp_get_num_teams [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()

4.0 omp_get_team_num [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()

4.0 omp_is_initial_device [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()

Lock Routines

General-purpose lock routines.

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*

Destroy lock [3.3.2] [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 [3.3.3] [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 [3.3.4] [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 [3.3.5] [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()

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.

4.0 [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.

4.0 [4.13] **OMP_DEFAULT_DEVICE** *device*
Sets the *default-device-var* ICV that controls the default device number to use in device constructs.

4.0 [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 adjustment the number of threads to use for executing **parallel** regions.

[4.9] [4.18] **OMP_MAX_ACTIVE_LEVELS** *levels*
Sets the *max-active-levels-var* ICV that controls the maximum number of nested active **parallel** regions.

[4.6] [4.5] **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.

4.0 [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 *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] [4.6] **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] [4.9] **OMP_THREAD_LIMIT** *limit*
Sets the *thread-limit-var* ICV that controls the number of threads participating in the OpenMP program.

[4.8] [4.7] **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 consume 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, 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**, **task**, or **teams** 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.

4.0 **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.

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)
.and.	(.true.)	**ior**	(0)
.or.	(.false.)	**ieor**	(0)
max (Least representable number in **reduction** list item type)			
min (Largest representable number in **reduction** list item type)			

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.

4.0 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.

(Continued >)

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.

4.0 SIMD Clauses [2.8.1, 2.8.2]

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(*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(*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.