An OpenMP executable directive applies to the succeeding structured block or an OpenMP construct. Each directive starts with `#pragma omp`. The remainder of the directive follows the conventions of the C and C++ standards for compiler directives. A structured-block is a single statement or a compound statement with a single entry at the top and a single exit at the bottom.

**parallel** [2.5] [2.5]
Forms a team of threads and starts parallel execution.

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

**structured-block**
clause:
- `if` (parallel: ) scalar-expression)
- num_threads(integer-expression)
- default(shared | none)
- private(list)
- firstprivate(list)
- shared(list)
- copyin(list)
- reduction(reduction-identifier: list)
- proc_bind(master | close | spread)

**for** [2.7.1] [2.7.1]
Specifies that the iterations of associated loops will be executed in parallel by threads in the team in the context of their implicit tasks.

```c
#pragma omp for [clause [ , clause] ...]
```

**for-loops**
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)
- nowait

**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 none remain.

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 from the run-sched-var IVC.

**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] [2.7.2]
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.

```c
#pragma omp sections [clause [ , clause] ...]
```

**structured-block**
clause:
- private(list)
- firstprivate(list)
- lastprivate(list)
- reduction(reduction-identifier: list)
- nowait

**single** [2.7.3] [2.7.3]
Specifies that the associated structured block is executed by only one of the threads in the team.

```c
#pragma omp single [clause [ , clause] ...]
```

**structured-block**
clause:
- private(list)
- firstprivate(list)
- lastprivate(list)
- copyin(list)
- reduction(reduction-identifier: list)
- nowait

**simd** [2.8.1] [2.8.1]
Applied to a loop to indicate that the loop can be transformed into a SIMD loop.

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

**for-loops**
clause:
- safelen(length)
- simulen(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]
Enables the creation of one or more versions that can process multiple arguments using SIMD instructions from a single invocation from a SIMD loop.

```c
#pragma omp declare simd [clause [ , clause] ...]
```

**[clause [ , clause] ...]**
function definition or declaration

clause:
- safelen(length)
- simulen(length)
- linear(list : linear-step)
- aligned(list : alignment)
- uniform(argument-list)
- inbranch
- notinbranch

**for simd** [2.8.3] [2.8.3]
Specifies that 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.

```c
#pragma omp for simd [clause [ , clause] ...]
```

**for-loops**
clause:
- Any accepted by the simd or 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.

```c
#pragma omp task [clause [ , clause] ...]
```

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

**taskloop** [2.9.2]
Specifies that the iterations of one or more associated loops will be executed in parallel using OpenMP tasks.

```c
#pragma omp taskloop [clause [ , clause] ...]
```

**for-loops**
clause:
- if( taskloop : ) scalar-expression)
- shared(list)
- private(list)
- firstprivate(list)
- lastprivate(list)
- default(shared | none)
- graniize(grain-size)
- num_tasks(num-tasks)
- collapse(n)
- final( scalar-expression)
- priority(priority-value)
- untied
- mergeable
- nogroup

Directives and Constructs for C/C++
Directives and Constructs for C/C++ (continued)

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.
#pragma omp taskloop simd [clause[ ... ]] for-loops
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.
#pragma omp taskyield

target data [2.10.1] [2.9.1]
Creates a device data environment for the extent of the region.
#pragma omp target data [clause[ ... ]] structured-block
clause: if([ target data : ] scalar-expression)
device(integer-expression)
map([ [map-type-modifier[ , ] map-type : ] map-type : ] list)
use_device_ptr(list)

target enter data [2.10.2]
Specifies that variables are mapped to a device data environment.
#pragma omp target enter data [clause[ ... ]] structured-block
clause: if([ target enter data : ] scalar-expression)
device(integer-expression)
map([ [map-type-modifier[ , ] map-type : ] map-type : ] list)
nowait

target exit data [2.10.3]
Specifies that list items are unmapped from a device data environment.
#pragma omp target exit data [clause[ ... ]] structured-block
clause: if([ target exit data : ] scalar-expression)
device(integer-expression)
map([ [map-type-modifier[ , ] map-type : ] map-type : ] list)
nowait

target teams [2.10.7] [2.9.5]
Creates a league of thread teams where the master thread of each team executes the region.
#pragma omp target teams [clause[ ... ]] structured-block
clause: num_teams(integer-expression)
thread_limit(integer-expression)
default(shared | none)
private(list)
firstprivate(list)
shared(list)
reduction(reduction-identifier : list)

distribute simd [2.10.9] [2.9.7]
Specifies loops which are executed concurrently using SIMD instructions.
#pragma omp distribute simd [clause[ ... ]] for-loops
clause: Any accepted by the distribute or simd directives.

#pragma omp 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.
#pragma omp target update clause[ ... ]
clause is motion-clause or one of:
if([ target update : ] scalar-expression)
device(integer-expression)
nowait
depend(dependence-type : list)
motion-clause:
from(list)
to(list)

#pragma omp declare target [2.10.6] [2.9.4]
A declarative directive that specifies that variables and functions are mapped to a device.
#pragma omp declare target declarations-definition-seq
#pragma omp end declare target

#pragma omp declare target [extended-list]
#pragma omp declare target clause[ ... ]
clause:
to(extended-list)
link(list)

teams [2.10.7] [2.9.5]
Creates a league of thread teams where the master thread of each team executes the region.
#pragma omp target teams [clause[ ... ]] structured-block
clause: num_teams(integer-expression)
thread_limit(integer-expression)
default(shared | none)
private(list)
firstprivate(list)
shared(list)
reduction(reduction-identifier : list)

distribute parallel for 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.
#pragma omp distribute parallel for simd [clause[ ... ]] for-loops
clause: Any accepted by the distribute or parallel for simd directives.

#pragma omp target parallel for [2.11.1] [2.10.1]
Shortcut for specifying a parallel construct containing one or more associated loops and no other statements.
#pragma omp target parallel for [clause[ ... ]] for-loop
clause: Any accepted by the parallel or for directives, except the nowait clause, with identical meanings and restrictions.

parallel sections [2.11.2] [2.10.2]
Shortcut for specifying a parallel construct containing one or more sections construct and no other statements.
#pragma omp parallel sections [clause[ ... ]] structured-block
#pragma omp parallel section
...
clause: Any accepted by the parallel or sections directives, except the nowait clause, with identical meanings and restrictions.

parallel for simd [2.11.4] [2.10.4]
Shortcut for specifying a parallel construct containing one simd construct and no other statements.
#pragma omp parallel for simd [clause[ ... ]] for-loops
clause: Any accepted by the parallel or for simd directives, except the nowait clause, with identical meanings and restrictions.

target parallel [2.11.5]
Shortcut for specifying a target construct containing a parallel construct and no other statements.
#pragma omp target parallel [clause[ ... ]] structured-block
clause: Any accepted by the target or parallel directives, except for copyin, with identical meanings and restrictions.

target parallel for [2.11.6]
Shortcut for specifying a target construct containing a parallel for construct and no other statements.
#pragma omp target parallel for [clause[ ... ]] for-loops
clause: Any accepted by the target or parallel for directives, except for copyin, with identical meanings and restrictions.

target parallel for simd [2.11.7]
Shortcut for specifying a target construct containing a parallel for simd construct and no other statements.
#pragma omp target parallel for simd [clause[ ... ]] for-loops
clause: Any accepted by the target or parallel for simd directives, except for copyin, with identical meanings and restrictions.
target simd [2.11.8]
Shortcut for specifying a target construct containing a
distribute simd construct and no other statements.

#pragma omp target simd [clause[, clause] ...]
for-loops
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
target teams construct and no other statements.

#pragma omp target teams [clause[, clause] ...]
structured-block
clause: Any accepted by the target or teams directives with identical meanings and restrictions.

target teams distribute [2.11.10] [2.10.6]
Shortcuts for specifying a teams construct containing a
target teams construct and no other statements.

#pragma omp target teams distribute [clause[, clause] ...]
for-loops
clause: Any accepted by the target or teams directives with identical meanings and restrictions.

target teams distribute simd [2.11.11] [2.10.7]
Shortcuts for specifying a teams construct containing a
target teams distribute simd construct and no other statements.

#pragma omp target teams distribute simd [clause[, clause] ...]
for-loops
clause: Any accepted by the target or teams distribute simd directives, with identical meanings and restrictions.

target teams distribute parallel [2.11.12] [2.10.8]
Shortcuts for specifying a target construct containing a
target teams distribute parallel construct and no other statements.

#pragma omp target teams distribute parallel [clause[, clause] ...]
for-loops
clause: Any accepted by the target or teams distribute parallel for directives.

target teams distribute parallel for [2.11.13] [2.10.9]
Shortcuts for specifying a target construct containing a
target teams distribute parallel for construct and no other statements.

#pragma omp target teams distribute parallel for [clause[, clause] ...]
for-loops
clause: Any accepted by the target or teams distribute parallel for or target

target teams distribute parallel for simd [2.11.14] [2.10.10]
Shortcuts for specifying a target construct containing a
target teams distribute parallel for simd construct and no other statements.

#pragma omp target teams distribute parallel for simd [clause[, clause] ...]
for-loops
clause: Any clause used for teams distribute parallel for or target

target teams distribute parallel for simd

#pragma omp target teams distribute parallel for simd [clause[, clause] ...]
for-loops
clause: Any clause used for teams distribute parallel for or target

target teams distribute parallel for simd

#pragma omp target teams distribute parallel for simd [clause[, clause] ...]
for-loops
clause: Any clause used for teams distribute parallel for or target

atomic (continued)
expression-stmt may be one of:

if atomic clause is... expression-stmt:
read
write
update or
is not present
{v = expr;}
{v = x; x = expr binop x;}
{v = x; x = expr binop x;}
{v = x; x = expr binop x;}
{v = x; x = expr binop x;}
{v = x; x = expr binop x;}
{v = x; x = expr binop x;}
{v = x; x = expr binop x;}
{v = x; x = expr binop x;}
{v = x; x = expr binop x;}
{v = x; x = expr binop x;}

flush [2.13.7] [2.12.7]
Executes the OpenMP flush operation, which makes a thread’s temporary view of memory consistent with
memory, and enforces an order on the memory operations of the variables.

#pragma 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.

#pragma omp ordered [clause[, clause] ...]
structured-block
clause:
threads
simd

#pragma omp ordered ordered-clause[[, clause] ...]

ordered-clause:
depend (source)
depend (sink : vec)

cancel [2.14.1] [2.13.1]
Requests cancellation of the innermost enclosing region of the
type specified. The cancel directive may not be used in
place of the statement following an if, while, do, switch,
or label.

#pragma omp cancel construct-type-clause[[, if-clause]
construct-type-clause:
parallel
sections
for
taskgroup
if-clause:
if (scalar-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.

#pragma omp cancellation point construct-type-clause
construct-type-clause:
parallel
sections
for
taskgroup

Continued
Directives and Constructs for C/C++ (continued)

threadprivate [3.2.10] [3.2.11] 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.

#pragma omp threadprivate(list) list: A comma-separated list of file-scope, namespace-scope, or static block-scope variables that do not have incomplete types.

declare reduction [3.2.14] [3.2.15] Declares a reduction-identifier that can be used in a reduction clause.

#pragma omp declare reduction(reduction-identifier: type-name-list; combiner: [initializer-clause] reduction-identifier: A base language identifier (for C), or an id-expression (for C++), or one of the following operators: +, -, *, &, |, ^, && and ||

type-name-list: A list of type names
combiner: An expression
initializer-clause: initializer (initializer-expr) where initializer-expr is omp_priv = initializer or function-name (argument-list)

Runtime Library Routines for C/C++

Execution environment routines affect and monitor threads, processors, and the parallel environment. The library routines are external functions with “C” linkage.

omp_get_nested [3.2.11] [3.2.12] Returns the value of the nest-var ICV, which indicates if nested parallelism is enabled or disabled.

int omp_get_nested(void);

omp_set_nested [3.2.12] [3.2.13] Affects the schedule that is applied when runtime is used as schedule kind, by setting the value of the run-sched-var ICV.

void omp_set_nested(void);

omp_get_num_threads [3.2.3] [3.2.4] 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.

int omp_get_num_threads(void);

omp_get_max_threads [3.2.15] [3.2.16] 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.

int omp_get_max_threads(void);

omp_get_thread_num [3.2.2] [3.2.3] Returns the thread number of the calling thread within the current team.

int omp_get_thread_num(void);

omp_get_num_procs [3.2.5] [3.2.6] Returns the number of processors that are available to the device at the time the routine is called.

int omp_get_num_procs(void);

omp_in_parallel [3.2.6] [3.2.7] Returns true if the active-levels-var ICV is greater than zero; otherwise it returns false.

int omp_in_parallel(void);

omp_set_dynamic [3.2.7] [3.2.8] 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.

void omp_set_dynamic(int dynamic_threads);

omp_get_dynamic [3.2.8] [3.2.9] 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.

int omp_get_dynamic(void);

omp_get_cancellation [3.2.9] [3.2.10] Returns the value of the cancel-var ICV, which is true if cancellation is activated; otherwise it returns false.

int omp_get_cancellation(void);

omp_set_cancellation [3.2.10] [3.2.11] Enables or disables nested parallelism, by setting the nest-var ICV.

void omp_set_cancellation(int nested);

omp_get_team_size [3.2.13] [3.2.14] 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.

int omp_get_team_size(int level);

omp_get_active_level [3.2.20] [3.2.21] 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.

int omp_get_active_level(void);

omp_in_final [3.2.21] [3.2.22] Returns true if the routine is executed in a final task region; otherwise, it returns false.

int omp_in_final(void);

omp_set_max_active_levels [3.2.22] [3.2.23] Returns the thread affinity policy to be used for the subsequent nested parallel regions that do not specify a proc_bind clause.

void omp_set_max_active_levels(int nthreads);

omp_get_num_places [3.2.23] [3.2.24] Returns the number of places available to the execution environment in the place list.

int omp_get_num_places(void);

omp_get_place_num_procs [3.2.24] [3.2.25] Returns the number of processors available to the execution environment in the specified place.

int omp_get_place_num_procs(int place_num);

omp_get_place_proc_ids [3.2.25] [3.2.26] Returns the numerical identifiers of the processors available to the execution environment in the specified place.

void omp_get_place_proc_ids(int place_num, int *id);

omp_get_partition_num_places [3.2.26] [3.2.27] Returns the number of places of the place to which the encountering thread is bound.

int omp_get_partition_num_places(void);

omp_get_partition_places [3.2.27] [3.2.28] Returns the number of places in the place partition of the innermost implicit task.

int omp_get_partition_places(void);

omp_get_partition_place_nums [3.2.28] [3.2.29] Returns the list of place numbers corresponding to the places in the place-partition-vars ICV of the innermost implicit task.

void omp_get_partition_place_nums(int *place_nums);
Runtime Library Routines for C/C++ (continued)

**Initialize lock with hint [3.3.2]**  
Initialize an OpenMP lock with a hint.

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

```c
void omp_init_nest_lock_with_hint(  
       omp_nest_lock_t *lock,  
       omp_nest_lock_hint_t hint);
```

**Destroy lock [3.3.3]**  
Ensure that the OpenMP lock is uninitialized.

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

```c
void omp_destroy_nest_lock(omp_nest_lock_t *lock);
```

**Set lock [3.3.4]**  
Sets an OpenMP lock. The calling task region is suspended until the lock is set.

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

```c
void omp_set_nest_lock(omp_nest_lock_t *lock);
```

**Unset lock [3.3.5]**  
Unsets an OpenMP lock.

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

```c
void omp_unset_nest_lock(omp_nest_lock_t *lock);
```

**Test lock [3.3.6]**  
Attempt to set an OpenMP lock but do not suspend execution of the task executing the routine.

```c
int omp_test_lock(omp_lock_t *lock);
```

```c
int omp_test_nest_lock(omp_nest_lock_t *lock);
```

**Timing Routines**
TIming routines support a portable wall clock timer. These record elapsed time per-thread and are not guaranteed to be globally consistent across all the threads participating in an application.

```c
omp_get_wtime(void);
```

```c
double omp_get_wtick(void);
```

**Device Memory Routines**
Timing routines support allocation and management of pointers in the data environments of target devices.

```c
void omp_target_alloc(size_t size, int dev_num);
```

**omp_target_free [3.5.2]**  
Frees the memory device allocated by the `omp_target_alloc` routine.

```c
void omp_target_free(void *ptr, int dev_num);
```

```c
int omp_target_is_present(void *ptr, int dev_num);
```

**omp_target_memcpy [3.5.4]**  
Copies memory between any combination of host and device pointers.

```c
int omp_target_memcpy(void *dst, void *src,  
                       size_t length, size_t dst_offset, size_t src_offset,  
                       int dst_dev_num, int src_dev_num);
```

**omp_target_memcpy_rect [3.5.5]**  
Copies a rectangular subvolume from a multi-dimensional array to another multi-dimensional array.

```c
int omp_target_memcpy_rect(void *dst, void *src,  
                           size_t length, size_t dst_element_size, size_t num_dims,  
                           const size_t *const size_t *const size_t *const size_t *const size_t *const size_t *,  
                           int dst_dev_num, int src_dev_num);
```

**omp_target_associate_ptr [3.5.6]**  
Maps a device pointer, which may be returned from `omp_target_alloc` or implementation-defined runtime routines, to a host pointer.

```c
int omp_target_associate_ptr(void *host_ptr,  
                             void *dev_ptr, size_t size_t *dev_offset,  
                             int dev_num);
```

**omp_target_disassociate_ptr [3.5.7]**  
Removes the associated pointer for a given device from a host pointer.

```c
int omp_target_disassociate_ptr(void *ptr,  
                                int dev_num);
```

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

**Initialize lock [3.3.1]**
Initialize an OpenMP lock.

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

```c
void omp_init_nest_lock(omp_nest_lock_t *lock);
```

Notes
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-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.

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.
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.

Data Sharing Attribute Clauses [2.15.3] [2.14.3]
Data-sharing attribute clauses apply only to variables whose names are visible in the construct on which the clause appears.

default(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]
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 [2.15.4] [2.14.4]
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.
copyprvate(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 [2.15.5] [2.14.5]
map[]([map-type-modifier][: map-type:list])
Maps a variable from the task’s data environment to the device data environment associated with the construct.
mapped-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-modifier:
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-exp)
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.

numtasks(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.

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>Implemented 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_set_bind()</td>
<td>Sec 4.4</td>
</tr>
<tr>
<td>stacksize-var</td>
<td>OMP_STACKSIZE</td>
<td>Implemented 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>Implemented 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>Implemented 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>Implemented 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>Implemented 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>